TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Course intro, Perfect Security, OTP, PRG

Lecturer: Elena Pagnin Lecture 1 on 5 Nov

Scribes: Lucia Lavagnino Last update November 22, 2024

Contents

1 Introduction

1.1 Cryptography: Meaning and Aims

Since the dawn of civilization humans have had the need for secrecy in conveying certain messages for varying reasons, from proceting resources, to gaining advantage in wars. "Cryptography" came from Ancient Greek: *kryptós* 'hidden, secret'; and *graphein*, 'to write', it is the practice and study of techniques for secure communication. Cryptography has woven itself into the fabric of human history, albeit with a somewhat understated presence.

In today's digital landscape, the mission of cryptography has evolved to "Make our Digital World Safe." Modern Cryptography covers a long array of properties including: confidentiality; data integrity; authenticity; entity identification; access control or authorisation; anonymity; non-repudiation; and privacy.

As security becomes a critical consideration in system design and development, professionals across various fields—system architects, software developers, and security specialists— must possess a robust understanding of security principles.

In this course, we will explore how cryptographic primitives and protocols are meticulously crafted so to guarantee security objectives, even in the presence of malicious actors. You will also practice identifying vulnerabilities in constructions and design flaws.

1.2 A very brief and abridged history of cryptography

- 1900 BC: The earliest recorded appearances of secret encondings belonged to the Egyptian monks. These monks developed a photocryptographic system using non-standard hieroglyphics to keep anyone outside of their inner circle from understanding what was being communicated;

- 100 BC: The Caesar cipher –named after the Roman general and statesman Gaius Julius Caesar– is one of the earliest and simplest forms of encryption techniques. It is a type of substitution cipher, which means each letter in the plaintext is replaced by a letter some fixed number of positions down or up the alphabet. In Caesar's case, a shift of three was commonly used. For example, 'A' would be replaced by 'D', 'B' would become 'E', and so on. The Caesar cipher is not secure by modern standards, it is vulnerable to frequency analysis and brute-force attacks.

- 1587: Mary Queen of Scots , sought to orchestrate the assassination of her cousin, Queen Elizabeth I, in order to claim the throne for herself. As part of this intricate scheme, she dispatched a coded letter to Sir Anthony Babington. The letter featured unusual symbols; however, it utilized a substitution cipher, where each symbol consistently represented the same letter. It did not take too long to the guards to figure out that the most common symbol was connected to the most common letter in the English language, and indeed,

all Mary's 'secret' communication was well monitored, decrypted, and became the proof of her treason. Consequently, Mary was executed.

- WW-II: In 1941, Mavis Batey was working on deciphering the 'uncophrehensible' stream of characters produced by Germans Enigma Machine (state of the art mechanical machines used to encrypt and decrypt secret messages). One day, Mary intercepted "a message with no Ls" in it. She figured out that the probabiltiy that a long palintext message contains no Ls at all was too low to be plausible, and conjectuerd that the cihpertext must be the encryption of a message made of only Ls (probably typed to test the machine). If this was true, it meant that the Enigma Machine would never encrypt a letter into itself. This arguably very small biased was enough to for Mavis' team to gain advantage in deciphering a subsequent message (they intercepted) which was planning an attack by Italy against the UK during the Battle of Cape Matapan. This decrypted message provided "invaluable information" that contributed to understanding and defeating the enemy's plans.

- Gene Grabeel (1920-2015) worked on deciphering intercepted Soviet ciphertext. During World War II, Russian militaries were using OTP. a perfectly secure encryption scheme as long as the encryption key is used only once. Due to the pressures from the German advance onto Moscow, the Soviet forces started to re-use encryption keys. The duplication –which undermines the security of the OTP– was discovered, and exploited by Gene's team to decrypt a large share secret messages which aided the Allies in winning the war.

If you want to read more about the use of cryptography in the history click here .

1.3 Kerckhoffs' Principle

The hystorical failures of cryptographic usage can be summarized into three issues:

- no rigorous security estimates/analysis;
- security by obscurity;
- bad security hygiene.

Auguste Kerckhoffs (19 January 1835 – 9 August 1903) was a Dutch linguist and cryptographer in the late 19th century. In his paper *La Cryptographie Militaire.* he surveyed the then state-of-the-art in military cryptography, and made a plea for considerable improvements, as well as practical advice and rules of thumb, including six principles of practical cipher design:

1. The system should be, if not theoretically unbreakable, unbreakable in practice.
2. The design of a system should not require secrecy, and compromise of the system should not inconvenience the correspondents (Kerckhoffs's principle).
3. The key should be memorable without notes and should be easily changeable.
4. The cryptograms should be transmittable by telegraph.
5. The apparatus or documents should be portable and operable by a single person.
6. The system should be easy, neither requiring knowledge of a long list of rules nor involving mental strain.

The Kerckhoffs' Principle is one of the foundations of cryptography.

Kerckhoffs' Principle (1884): A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

In Clause Shannon's words: the enemy knows the system, so we should design systems under the assumption that the enemy will immediately gain full familiarity with them, i.e. *no security via obscurity.*

Why is it OK to assume the key is secret and not the algorithm? Because it is always possible to choose a fresh key.

1.4 Foundation of Modern Cryptography

In a modern cryptography (meaning cryptography from the 1980s till now), first, we need to find rigorous definitions: such as What does security mean? What are the attacker's goal and resources? Precise mathematical security assumptions (formally define "hard")? And then, we want to establish a rigorous logical reasoning to argue security.
The field of Cryptanalysis, instead, uses many heuristics to define exact security levels, and solutions need to work in practice. This means efficient algorithms and using the lowest ratio size/security.

Combined together cryptography and cryptanalysis form the field of so-called Cryptology.

In this course, we are mainly concerned with cryptography and with answer the question: *What does provable security mean?* In order to answer to this question, we need some tools:

- Providing mathematical *definitions of security* (e.g. through "security games" between algorithms and probabilities to quantify the chance of "winning")
- Providing precise *mathematical assumptions* (e.g. "the discrete logarithm problem is hard", where hard is formally defined).
- Providing *proofs of security* (often reasoning by **reduction**: if a scheme can be broken, then there is a way to contradict a mathematical assumption. Hence, if we believe in the mathematical assumption, the scheme cannot be broken).

Figure 1: Scheme of how reduction works

2 Symmetric Encryption

2.1 Secure Communication

The paranoia world of cryptographers, lets them constantly anticipate the worst-case scenarios and design systems that ensure secure communication even in the presence of potential adversaries. Their guiding principle is simple: trust no one but the intended recipient. Before moving forward, we should introduce the cast of characters commonly employed in cryptographic discussions: **Alice**, **Bob**, and **Eve**. These characters were first introduced by cryptographer Ron Rivest in his paper "Methods for Obtaining Digital Signatures and Public-Key Cryptosystems" in 1978. Alice is usually the party who initiates communication and sends a message to Bob. Bob, on the other hand, receives the message sent by Alice and is the intended recipient. Eve, the eavesdropper, tries to intercept the communication between Alice and Bob to gain unauthorized access to the information being exchanged. One reason for the popularity of Alice, Bob, and Eve in cryptography is their gender-neutral names. This allows for inclusive language in technical discussions and makes it easier for a diverse range of individuals to relate to the scenarios being presented. Additionally, the simplicity of the names makes it easier for beginners to grasp cryptographic concepts without getting bogged down by technical jargon.

Cryptographic primitives and protocols are designed to maintain a desired set of security goals even under attempts at making them deviate from the expected functionality.

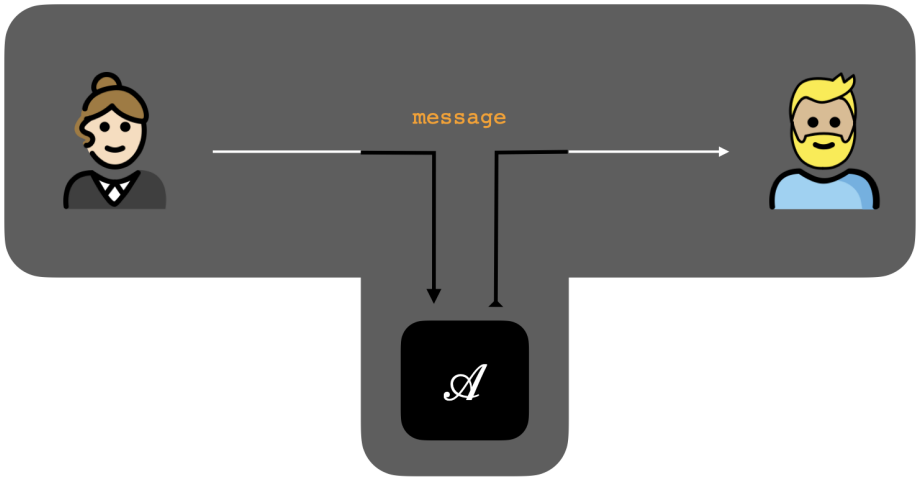


Figure 2: Insecure comunication, the message from Alice to Bob is seen by the malicious adversary $\mathcal{A}$

In order to build secure communication over an insecure channel, we need first to understand what we mean by security. To do so, we need to define what the adversary is after, what it can or cannot do... for now let us settle with:

- What's the adversary's goal? To learn the message.
- What are resources? Eavesdrop the channel (observe the message exchanges)
- What does secure mean? [more on this in a second]
- What can Alice and Bob use? Symmetric encryption

Many of the troubles that cryptosystem designers faced over history (and still face!) can be attributed to not properly defining or understanding what the goals they want to achieve are in the first place.

2.2 Symmetric Encryption

Definition 1 (**Symmetric Encryption scheme**). *A tuple* $(KeyGen, E, D)$ *is a* ***symmetric encryption scheme*** *over the sets* $\mathcal{K}$ *(key space),* $\mathcal{M}$ *(message space), and* $\mathcal{C}$ *(ciphertext space) if all algorithms are efficient and satisfy the following:*

$KeyGen(1^n) \to k$ *the key generation is a randomised algorithm that returns a key . (This algorithm is often implicit and assumed to be* $k \xleftarrow{\$} \mathcal{K}$ *).*

$E(k, m) \to c$ *the encryption is a possibly randomised algorithm that on input a key and a (plaintext) message* m*, outputs a ciphertext* c*.*

$D(k, m) \to m$ *the decryption is a deterministic algorithm that on input a key and ciphertext, outputs a plaintext message.*

Moreover, a Symmetric Encryption Scheme $(KeyGen, E, D)$ *should satisfy the* ***correctness****, i.e.* $\forall m \in \mathcal{M}$*, it should hold that*

$$Pr\left[D(k, E(k, m)) = m | k \xleftarrow{\$} KeyGen(1^n)\right] = 1.$$

The last formula means that if we pick at random a key $k \xleftarrow{\$} KeyGen(1^n)$ and we compute the ciphertext and plaintext in the right way, the algorithms should give us back exactly the original message, so with probability equal to one we obtain that $D(k, E(k, m)) = m$.

In figure 3, there is a visualisation of how an symmetric encryption scheme works.

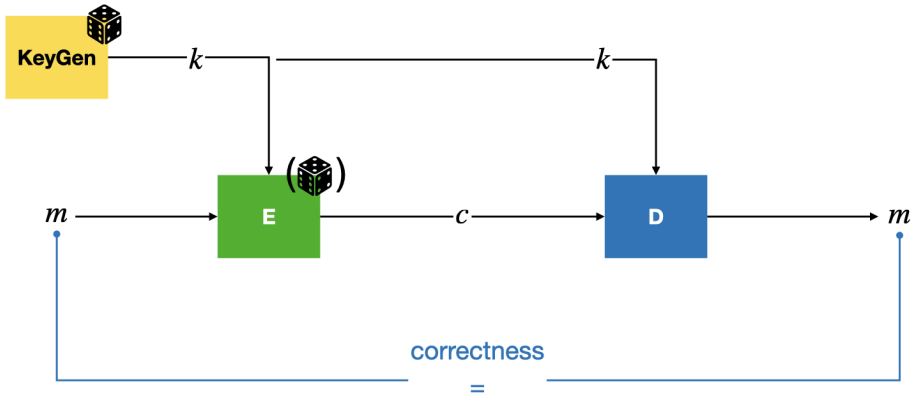


Figure 3: Visualization of Symmetric Encryption

2.3 The One Time Pad (OTP)

An example of Symmetric Encryption scheme is given by the One-Time Pad (OTP). In OTP $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^n$. The encryption is exceedingly simple: to encrypt a message $m \in \mathcal{M}$ with a key $k \in \mathcal{K}$ we simply output $m \oplus k$ where $\oplus$ is the bitwise XOR operations that outputs the string corresponding to XORing each coordinate of m and k. The decryption works in the same way: to decrypt a ciphertext $c \in \mathcal{C}$ with a key $k \in \mathcal{K}$ we simply output $c \oplus k$.

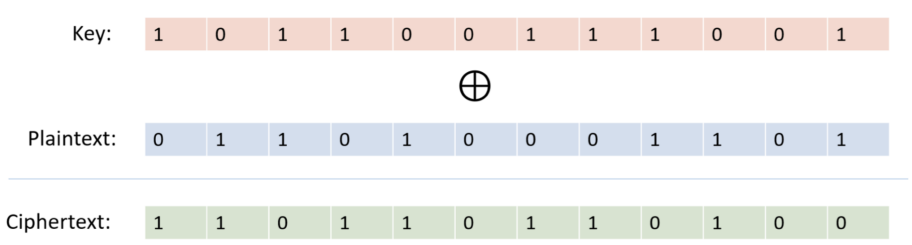


Figure 4: Example of encryption with OTP

What does *secure* mean? Intuitively, the adversary $\mathcal{A}$ should not learn the message. This means that the ciphertext c should not leak any information about the message, a part from the length of m. The key insight of Shannon was that in a secure encryption scheme the ciphertext should not reveal any additional information about the plaintext (a part from its length).

2.4 Perfect Secrecy

Definition 2 (**Perfect Secrecy (Perfect Security)**). *A symmetric encryption scheme* $(KeyGen, E, D)$ *is* ***perfectly secret*** *if for all pair of messages* $m_0, m_1 \in \mathcal{M}$ *and for all ciphertexts* c *it holds that:*

$$Pr\left[c \leftarrow E(k, m_0) | k \leftarrow KeyGen(1^n)\right] = Pr\left[c \leftarrow E(k, m_0) | k \leftarrow KeyGen(1^n)\right]$$

In other words, the probability that Eve guessed the plaintext, should not change after seeing the ciphertext. Security notions essentially state that for every pair of messages, the probabilities that either message maps to a given ciphertext c must be equal.

2.5 Shannon's Theorem

Theorem 1 (Shannon 1949). *The One-Time Pad is a perfectly secure private-key encryption scheme.*

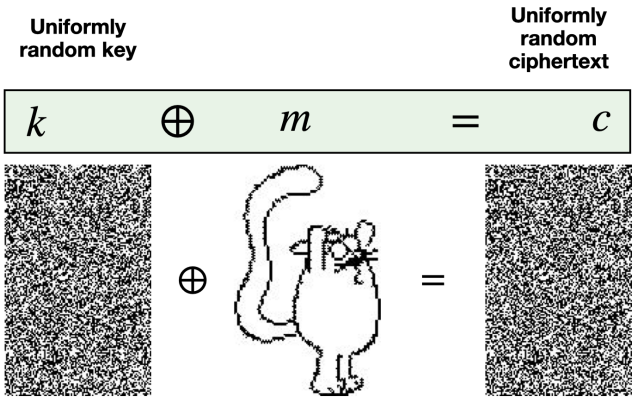


Figure 5: OTP is perfectly secure

Proof. First, show that OTP is a symmetric encryption scheme. To do so, define $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^n$, and the algorithms $KeyGen(1^n) : k \xleftarrow{\$} \{0,1\}^n$, $E(k,m) = k \oplus m$ and $D(k,c) = k \oplus c$.
Correctness holds since for all k, and all m:

$$D(k, E(k,m)) = k \oplus c = k \oplus (k \oplus m) = m.$$

Next, notice that for every message m and ciphertext c there is exactly one key $k \in \{0,1\}^n$ such that $c = E(k,m)$ (the key is $k = m \oplus c$). Then:

$$Pr[c \leftarrow E(k,m) | k \leftarrow KeyGen(1^n)] = Pr[c \oplus m \leftarrow KeyGen(1^n)] = \frac{1}{|\mathcal{K}|}$$

Since this holds for every message m, we have that:

$$Pr[E(k,m_0) \rightarrow c | k \leftarrow KeyGen(1^n)] = \frac{1}{|\mathcal{K}|} = Pr[E(k,m_1) \rightarrow c | k \leftarrow KeyGen(1^n)]$$

□

Perfect-secrecy is a form of **information-theoretic** (aka unconditional) security notion, that means: the security property holds no matter what computational power, time-, or space-limit the adversary has.

Computational security is based on the conjectured gap between the complexity classes P and NP, that means: we have efficient algorithms for the legitimate user, and inefficient algorithm for the adversary. A **gap** is a hard problem (can be verified quickly but no efficient algorithm is known to solved it). What is computationally hard depends on the adversary's resources (classical vs quantum computing, storage, time..).

Computational solutions are usually more efficient and practical than information-theoretic ones BUT they always have a "best before date". The P versus NP problem is a major unsolved problem in theoretical computer science. In informal terms, it asks whether every problem whose solution can be quickly verified can also be quickly solved.

The OTP encryption scheme has several drawbacks:

- The key is as long as the message.
- The key should be used only to encrypt ONE message.

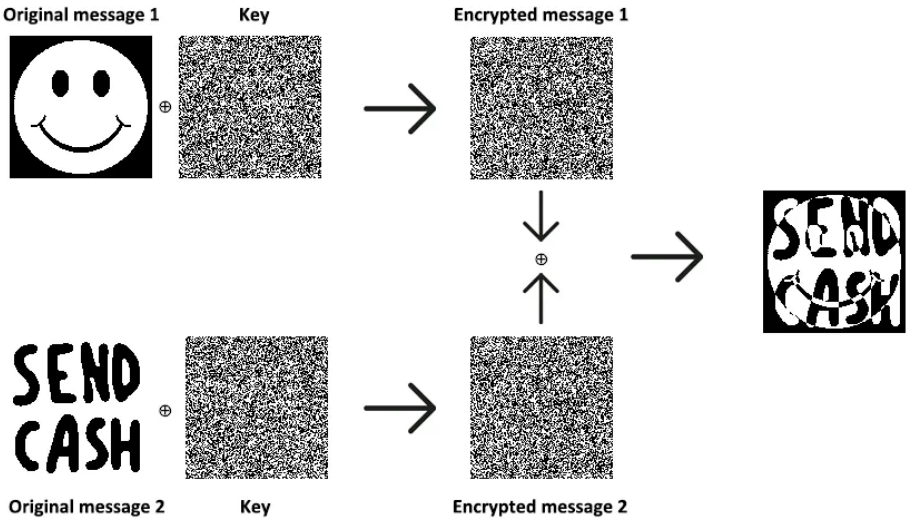


Figure 6: If we use the same key twice, we can see patterns in the ciphertext.

Assume that the same key is used twice to produce $c_0 = k \oplus m_0$ and $c_0 = k \oplus m_0$. Assume the adversary $\mathcal{A}$ gets hold of the two ciphertexts, then $\mathcal{A}$ can compute:

$$c_0 \oplus c_1 = (k \oplus m_0) \oplus (k \oplus m_1) = m_0 \oplus m_1.$$

But $m_0 \oplus m_1$ conveys a lot of information about m_0 and m_1. This is not acceptable.

- The ciphertext is (intentionally!) malleable.

Theorem 2 (Shannon 1940s)**.** *If a symmetric encryption scheme* $(KeyGen, E, D)$ *defined over* $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ *has perfect security then* $|\mathcal{K}| \geq |\mathcal{M}|$.

Proof. Fix arbitrary $m_0 \in \mathcal{M}$ and $k_0 \in \mathcal{K}$, and let $c_0 = E(k_0, m_0)$. Since the encryption scheme has perfect secrecy, it holds that "every message has the same probability to be encrypted to c", i.e., $Pr[c_0 = E(k, m)] = Pr[c_0 = E(k, m_0)] > 0$. Thus for each $m_i \in \mathcal{M}$ there is a key $k_i \in \mathcal{K}$ such that $E(k_i, m_i) = c_0$. But each message must have a different key! If the same key k would map two plaintexts m_0 and m_1 to the same ciphertext $c_0 = E(k, m_1) = E(k, m_0)$, then we lose correctness (the decryption of c_0 for the key k becomes ambiguous). Thus there must be at least as many keys as messages: $|\mathcal{K}| \geq |\mathcal{M}|$.

□

Note: perfect security is impractical. How close to perfect security can we go, while being practical? The core of crypto is randomness. The perfect secrecy of OTP comes from using one random key to mask/hide one message. Since we cannot reuse the key (otherwise we lose security), can we 'expand' it to get new keys? The solution is given by randomness.

TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

PRG, Block Ciphers, Modes of Operation

Lecturer: Elena Pagnin Lecture 2 on 8 Nov

Scribes: Lucia Lavagnino Last update December 2, 2024

Contents

1 Pseudorandom Generators (PRG)

1.1 Definition

Definition 1. *A **Pseudo Random Generator** is a deterministic function* $\mathsf{PRG} : \{0,1\}^n \to \{0,1\}^\ell$ *such that:*

1. $\mathsf{PRG}(\cdot)$ *is efficiently computable;*
2. $\mathsf{PRG}(\cdot)$ *expands its input, i.e.* $\ell > n$*;*
3. *The ensemble* $\{\mathsf{PRG}(x) | x \xleftarrow{\$} \{0,1\}^n\}$ *is pseudo-random, i.e., for (polynomially many random sampled inputs)* $x \xleftarrow{\$} \{0,1\}^n$ *no efficient adversary can tell apart the output* $y = \mathsf{PRG}(x) \in \{0,1\}^\ell$ *from a (collection of polynomially many) random string* $z \xleftarrow{\$} \{0,1\}^\ell$*.*

The third condition in the definition asks that the output of the PRG *appears* random. We can not ask its output to be purely random for several reasons, for instance, the PRG can output at most 2^n different ℓ-bit strings (one y per input $x \in \{0,1\}^n$), and these do not cover all the possible strings in the set of destination of PRG (there are 2^ℓ many string in $\{0,1\}^\ell$). Hence, the best we can require from a pseudo random generator function is that when evaluated on m many random inputs, the PRG outputs are indistinguishable from m many strings drawn uniformly at random from $\{0,1\}^\ell$. We should also clarify *to whom* these two sets should appear indistinguishable. In cryptography, the entity in question is always the adversary, which is modeled as an algorithm that runs in time polynomial in n (also known as computationally bounded).

The best way to check if a candidate algorithm is a PRG is by running a series of tests, there is **no mathematical proof**! We can reason about the security of a PRG using a formal (mathematical) security game.

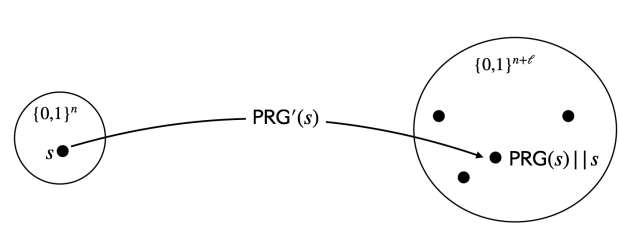
Example 1. *(**The RSA** PRG) Let p, q be two n-bit long primes, $N = p \cdot q$, and $e \in \mathbb{Z}^*_{\varphi(N)}$ a random invertible element. Let $LSB(x)$ be the function that returns the least significant bit of its input $x \in \mathbb{Z}^*_N$. Then $\mathsf{PRG}_{RSA} : \{0,1\}^n \to \{0,1\}^\ell$ is defined as*

$$\mathsf{PRG}_{RSA}(x) = LSB(x)||LSB(x^e \mod N)||\cdots||LSB(x^{e^\ell})$$

For a concrete example, take $N = 531761$, $e = 493589$ and $x = 72979$; then $\mathsf{PRG}_{RSA}(x) = [1,1,0,0,0,0,1,0,1,1,0]$. Note that we can not stretch this sequence forever: at a certain point the digits will loop back because we have finite many numbers modulo N. The idea is that if we start from a small x value, we can continue a long time until they loop back, and until they loop back, the sequence of 0s and 1s is pseudo-random.

Aside strings generated by a PRG are far from random! See Example 2.

Example 2. *Consider the following candidate for a pseudorandom function defined as $\mathsf{PRG}'(s) = \mathsf{PRG}(s)||s$ from a pseudorandom generator* PRG*.*



PRG' *outputs strings of $n + l$ bits, but actually the size of its image is as large as its input so no more than 2^n strings can be generated by* PRG'*.*
This is clearly not random, we are just sampling a little space. The idea is that understanding which little space we are mapping should be hard for an adversary $\mathcal{A}$ to understand.

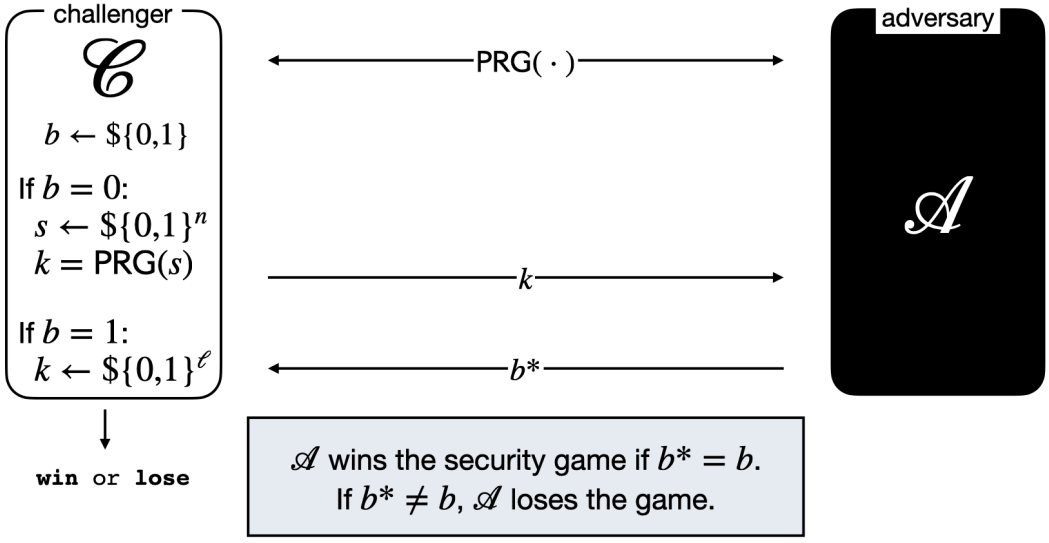
1.2 Security

Intuition: can you distinguish a random-looking string from a purely random string? We know that PRG can not be exactly uniform, but how close can it be?

Modeling security In modern cryptography security for a cryptographic scheme is often modeled as game. A security game is nothing more than a well-codified connection between two algorithms: the **challenger** (for which we specify the behavior/code according to the security notion we want to capture) and the **adversary** (for which we do not know the code, and hence it is treated as a black box). If, after the interaction, the adversary outputs a correct answer to the challenge it received during the game, we say that the adversary **wins**, and the scheme under examination is deemed to be **insecure**. On the other hand, if no adversary can win the game, then the scheme is secure. Unless otherwise specified, we will always assume the adversary is **efficient**, that is to say, it can sample randomness (it is a probabilistic algorithm) and it runs in polynomial time (hence the amount of computations it can do is bounded). In this case, security is defined as the probability that the adversary wins the game, and it is often enough to show that this probability is very very small, but not necessarily 0 (more on this in Section 1.3)

The PRG security game The aim is to quantify the attacker's likelihood in distinguishing PRG from a source of uniform randomness over $\{0,1\}^\ell$. In this context, uniform means that all the strings have the same probability of being sampled.

Let us see our first example of security game: the PRG security game. This game is designed so that if the adversary can tell the difference between PRG outptus and purely random strings, then it wins the game and the PRG is not secure. We provide a verbose description of the PRG security game that is depicted in the figure below.



The game starts by distributing to the challenger and the adversary the description of the PRG function. This is according to Kerckhoffs's principle (no security by obscurity, hence everyone knows the cryptographic algorithms). Next, the game follows these steps

1. The challenger (denoted by a calligraphic capital C letter, $\mathcal{C}$) draws a uniformly random bit $b \xleftarrow{\$} \{0,1\}$.
   If $b = 0$, the challenger draws a random seed $s \xleftarrow{\$} \{0,1\}^n$ and computes $k = \mathsf{PRG}(s)$.
   Else ($b = 1$), the challenger draws a uniformly random string $k \xleftarrow{\$} \{0,1\}^\ell$.
2. $\mathcal{C}$ sends k to $\mathcal{A}$.
3. $\mathcal{A}$ tries to determine b from k.
4. $\mathcal{A}$ sends its guess b^* to $\mathcal{C}$ (within polynomial time)
5. $\mathcal{C}$ outputs **win** if $b^* = b$, else it outputs **lose**.

Security statement A pseudo random function $\mathsf{PRG} : \{0,1\}^n \to \{0,1\}^\ell$ is a **secure** PRG if any probabilistic polynomial time attacker $\mathcal{A}$ has only negligible advantage in winning the PRG security game. Formally,

$$Adv(\mathcal{A}) = \left| Pr[\mathcal{A}\ wins] - \frac{1}{2} \right| < negl(n)$$

Observation: we have $-\frac{1}{2}$ in the previous formula because we know that, with probability $\frac{1}{2}$, the bit is either 0 or 1, so picking at random a bit the adversary $\mathcal{A}$ will be correct $\frac{1}{2}$ of the times. With the formulas, we want to say that the probability that the adversary $\mathcal{A}$ is not far away from the probability of randomly guessing.

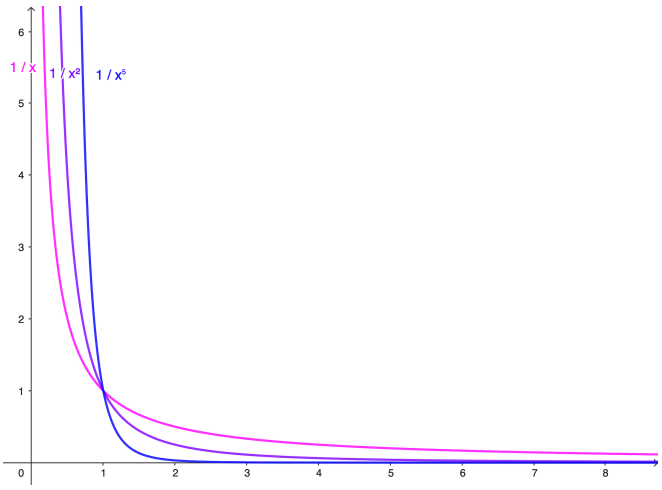
1.3 Negligible

Most of the security statements we see in the course will bound the adversary's winning probability by a *negligible* function. But what does negligible mean? In simple terms it means "too small to matter".

Definition 2. *In order to be able to formally reason about security, we need to formalize this intuition in mathematical terms. A function* $negl(x) : \mathbb{N} \to \mathbb{R}_{\geq 0}$ *is negligible in* x *if for any positive polynomial* $p(x)$ *it holds that*

$$negl(x) \leq \frac{1}{p(x)}$$

for all $x \geq x_0 \in \mathbb{N}$.



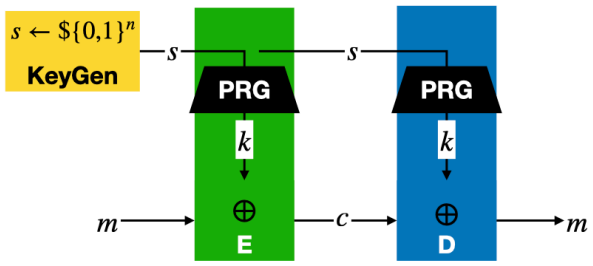
The definition asserts that there is a point, after which, my *negl* function is smaller than all the inverse of polynomials. In the picture, we can see that the definition translates "too small to matter", in mathematical terms.

Intuition: Events that occur with negligible probability occur so seldom that polynomial time algorithms will never see them happening. This definition is asymptotic ("it holds from a certain point onwards"). This is a common approach in complexity-based cryptography. In practice, if one needs to pick a value, then $negl(x) < 2^{-128}$ is considered to be negligible (but this depends on the context, and may yield inefficient constructions). To have an idea of how small the number is, if you consider the probability of guessing the favourite song that a random person is listening to on Spotify, this probability is much greater than $negl(x) < 2^{-128}$.

1.4 Secure Encryption from PRG

Now that we have a new tool (PRG), we can use it to create an encryption scheme, that is hopefully better than the OTP. Let $\mathsf{PRG} : \{0,1\}^n \to \{0,1\}^\ell$ be a secure pseudo random generator. We can built a One-Time PRG cipher by defining:

- $\mathcal{M} = \mathcal{C} = \{0,1\}^\ell$, $\mathcal{K} = \{0,1\}^n$, $n < \ell$
- $KeyGen(1^n) \to s$ (where $s \xleftarrow{\$} \{0,1\}^n$)
- $Enc(s, m) = \mathsf{PRG}(s) \oplus m$
- $Dec(s, c) = \mathsf{PRG}(s) \oplus c$



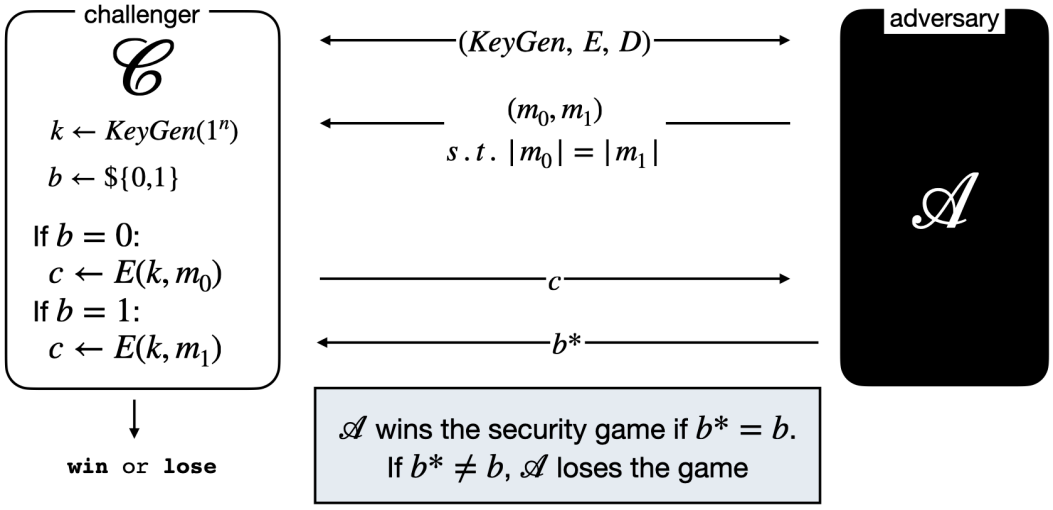
The evaluation on the PRG , allows us to construct a longer key whose size matches the length of the message.

Is the One-Time PRG cipher perfectly secure? No, the One-Time PRG cipher cannot be perfectly secure because it does not satisfy the Shannon's Theorem (which states that perfect secrecy implies the key space be at least as large as the message space). Hence we need to introduce a new security definition that works for ciphers with shorter keys $|\mathcal{K}| < |\mathcal{M}|$.

1.5 Semantic Security

Perfect secrecy says that given a cipher text all messages are equal probable preimages of that cipher text. Now we are relaxing this definition by saying that the adversary $\mathcal{A}$ should not be allowed to distinguish if the ciphertext comes from a specific m_0 or m_1. The two definitions are similar but not the same and the little gap between them is what enables us to have efficient encryption that we could not have with OTP.
Aim: quantify the attacker's likelihood in distinguishing an encryption of a (chosen) message m_0 from an encryption of another (chosen) message m_1.

Intuitively the semantic security game for an encryption scheme is described in the following picture:



Verbose description of the semantic security game

1. The challenger $\mathcal{C}$ generates a key $k \leftarrow KeyGen(1^n)$ and draws a random bit $b \xleftarrow{\$} \{0,1\}$.
2. The adversary $\mathcal{A}$ chooses two messages m_0, m_1 of the same length and sends them to $\mathcal{C}$.

3. $\mathcal{C}$ encrypts m_b according to the bit drawn in step 1, and returns $c = Enc(k, m_b)$ to $\mathcal{A}$.

4. $\mathcal{A}$ tries to determine b from c, m_0 and m_1; and eventually sends its guess b^* to the $\mathcal{C}$.

5. The challenger outputs win if $b^* = b$; else it outputs lose.

Definition 3. *A symmetric encryption scheme is* ***semantically secure*** *if any efficient attacker $\mathcal{A}$ has only negligible advantage in winning the semantic security game. Formally,*

$$Adv(\mathcal{A}) = \left| Pr[\mathcal{A} \text{ wins}] - \frac{1}{2} \right| < negl(n)$$

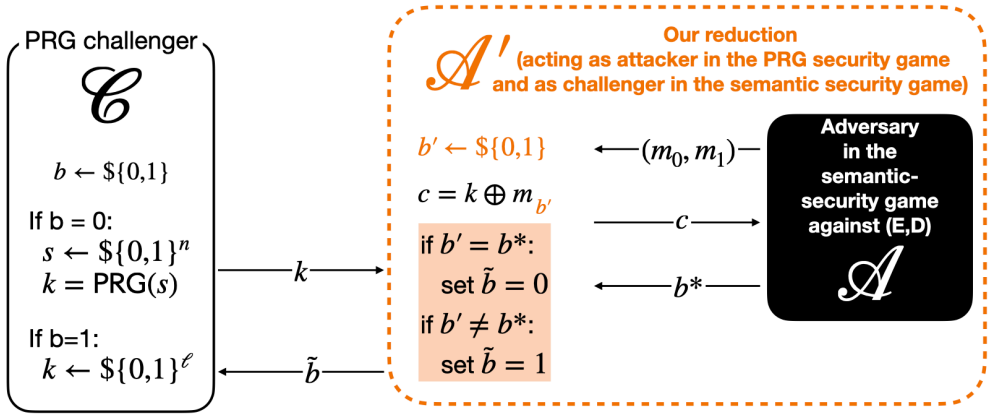
Remarks on the definition:

- We don't expect the encryption scheme to hide the length of the plaintext; (hence m_0 and m_1 must have the same length).
- An attacker who just guesses, choosing a random $b^* \xleftarrow{\$} \{0,1\}$, has advantage 0. An attacker who always answers $b^* = 1$ (or $b^* = 0$) also has advantage 0.
- If the encryption scheme is the OTP, any attacker $\mathcal{A}$ has advantage 0 because there is a key that explain both m_0 and m_1. Hence the OTP is both perfectly and semantically secure.

Theorem 1. *If* $\mathsf{PRG} : \{0,1\}^n \to \{0,1\}^\ell$ *is a secure* PRG*, then the cipher defined by* $Enc(s, m) = \mathsf{PRG}(s) \oplus m$ $Dec(s, c) = \mathsf{PRG}(s) \oplus c$ *is semantically secure.*

Proof. We must prove that any efficient adversary against the encryption's semantical security has negligible advantage, without knowing anything about the adversary's strategy. We use the proof technique "*reduction to absurd*". Idea of the proof: assume that there exists an adversary $\mathcal{A}$ that can break the semantic security of the encryption; then we build a new adversary $\mathcal{A}$' that uses $\mathcal{A}$ to break the security of the PRG ; since PRG is assumed to be secure, such $\mathcal{A}'$ cannot exist. Thus it was absurd to assume $\mathcal{A}$ exists in the first place.

This following part is not required in the exam.



Important observations:

If $b = 0$, the ciphertext c is the encryption using the PRG cipher. Because we assumed that $\mathcal{A}$ wins this game with non negligible probability this means $b' = b^*$. So $\mathcal{A}$' wins when $\mathcal{A}$ does. If $b = 1$, $\mathcal{A}$' encryption is the OTP (perfectly secure), thus $\mathcal{A}$ has no advantage. So $\mathcal{A}$' only guesses correctly with 1/2 probability (0 advantage).

Using the definition of conditional probability

$$Pr[A|B] = \frac{Pr[A \cap B]}{Pr[B]}$$

we have that

$$\begin{aligned} Pr[\mathcal{A}' \text{ wins } \mathsf{PRG}] &= Pr[\mathcal{A}' \text{ wins AND } b=0] + Pr[\mathcal{A}' \text{ wins AND } b=1] = \\ &= Pr[\mathcal{A}' \text{ wins } |b=0]Pr[b=0] + Pr[\mathcal{A}' \text{ wins } |b=1]Pr[b=1] = \\ &= Pr[\mathcal{A} \text{ wins sem.sec}] \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \end{aligned}$$

Or, reorganising the terms:

$$Pr[\mathcal{A} \text{ wins sem.sec}] = 2Pr[\mathcal{A}' \text{ wins } \mathsf{PRG}] - \frac{1}{2}$$

hence

$$Adv_{sem.sec}(\mathcal{A}) = |Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}| = |(2Pr[\mathcal{A}' \text{ wins } \mathsf{PRG}] - 1/2) - \frac{1}{2}| = 2 \cdot Adv_{\mathsf{PRG}}(\mathcal{A}')$$

□

This is because if our PRG -based encryption is not secure then $\mathcal{A}$ has a non-negligible advantage in winning the semantic security game. If that was the case, we have constructed an efficient (PPT) reduction/adversary $\mathcal{A}$' that uses $\mathcal{A}$ to win the PRG security game and has twice the advantage of $\mathcal{A}$. Since we assumed the PRG to be secure, it is impossible for any efficient adversary to break the PRG . So such an $\mathcal{A}$' cannot exist. Which in turn implies that $\mathcal{A}$ cannot exist. So it was absurd to assume such an $\mathcal{A}$ exists. This reasoning implies that our PRG -based encryption is provably secure.

2 Block Ciphers

2.1 Pseudorandom Permutation

Up until this point, our security games model encryption only one message (either m_0 or m_1), but in practice we would like to be able to reuse the same (short) key to encrypt several messages. We are after a new definition of security that holds for encrypting multiple messages, and messages of potentially different lengths.

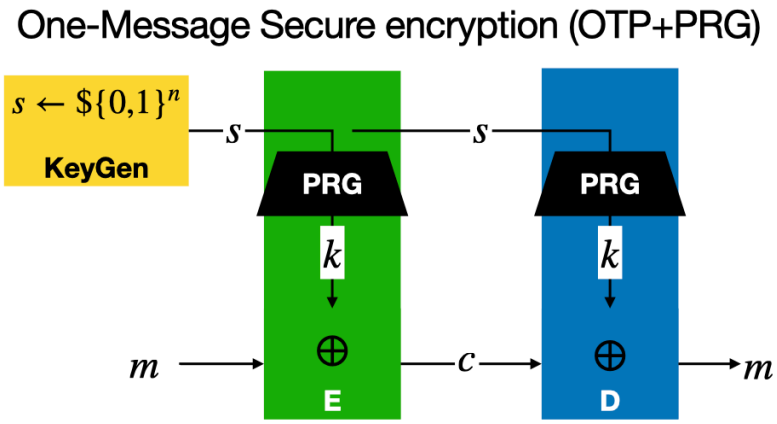
We would like to obtain secure and efficient encryption. There are three principles:

Reuse : we would like to have one secret key to do many encryptions, we want to encrypt polynomial many messages;

Reduce : we cannot have the length of the OTP to be as long as the number of messages we want to ship. It needs to be short;

Recycle : could we evolve a key? Could we produce a kind of randomness keys and recycle it?

A possible candidate for Multi-Message Secure Encryption could be the one described in the following way:



- Define encryption $E(s,m) = (s||\mathsf{PRG}(s) \oplus m) = c$ for a random $s \xleftarrow{\$} \{0,1\}^n$, different for every message. The problem is that we need also to share the value s, but given s, everybody can decrypt.
- To solve the problem, we need to find a way to make the PRG depend on a shared key. This object is called Pseudo Random Permutation $f_k : \{0,1\}^n \to \{0,1\}^n$. Encryption is defined as $c = (s||f_k(s) \oplus m)$ for a random s (different for every message).

A **random permutation** is a permutation chosen uniformly at random from the set $Perms(\mathcal{M})$ of all permutations (one-to-one maps) on the set $\mathcal{M}$.

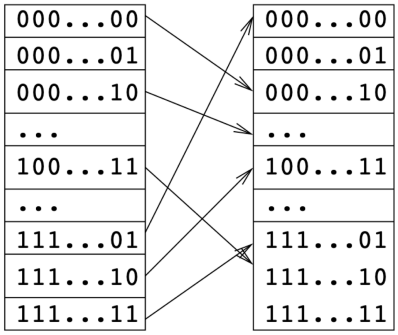


Figure 1: How a random permutation looks like

Informal: A family of pseudo-random permutations PRP $\{f_k : \{0,1\}^n \to \{0,1\}^n\}_{k \in \{0,1\}^\kappa}$ is a collection of permutations such that:

- each f_k can be efficiently computed; and
- there is no efficient way to distinguish between f_k and a random function.

If we describe a purely random function, there is no compact way to describe it, we need to list all the inputs and outputs. We need a compact way to describe f_k that allows us to produce something that looks random, and rearrange the string, but can be represented in an efficient way to be computed in polynomial time.
Over $\mathcal{M} = \{0,1\}^n$, there are

$$|Perm(\mathcal{M})| = 2^n! \approx 2^{n2^n}$$
$$|PRP| = |0,1^K| = 2^K$$

where K is the size of the $\mathcal{K}$.
The number of permutations is $2^n! \approx 2^{n2^n}$, an enormous number. In contrast, the number of block ciphers is just 2^k, a negligible share. A description of a permutation has exponential length ($n \cdot 2n$ bits). So an efficient attacker who gets a permutation as an input doesn't even have time to read his input.

Theorem 2. *If f_k is a PRP then the scheme $(KeyGen, E, D)$ where $c = (s||f_k(s) \oplus m)$ is a secure encryption scheme.*

We do not discuss the proof of this theorem in the class, but it is a very good exercise for crypto afictionados.

2.2 Definition of Block Cipher

Definition 4. *A **Block Cipher** is a keyed function that is deterministic and invertible. Formally, $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$, where $\mathcal{K} = \{0,1\}^K$ is a key space, $\mathcal{M} = \{0,1\}^n$is the block space and for every $k \in \mathcal{K}$, the function $E(k,\cdot) : \{0,1\}^n \to \{0,1\}^n$ is invertible, that is to say, there exist an efficient function $D(k,\cdot) : \{0,1\}^n \to \{0,1\}^n$ such that $D(k, E(k,m)) = m$ for every $m \in \{0,1\}^n$. Plaintexts and ciphertexts are both called **blocks**.*

In the definition, by a keyed function, we mean a function completely defined by a key-string. It is important that the keyed function is invertible because we need to be able to decrypt.
The Block cipher "chains" blocks according to a "mode of operation" (more on this later). Observation: we can now reuse the same key to encrypt/decrypt multiple messages! Typical values today are $|\mathcal{K}|$ equal to 2^{128} or 2^{256}. We cannot (efficiently) implement (all possible) random permutations for reasonable sizes of n. Instead, we strive to construct ciphers that cannot be distinguished from random permutations.

2.3 Chosen Plaintext Attack (IND-CPA)

Now we can encrypt multiple messages. Considering that the adversary $\mathcal{A}$ can see a bunch of messages encrypted, we want to quantify the probability of the encryption. The term *adaptive queries* in figure 2 is used to specify that the adversary $\mathcal{A}$ can choose a message, then see the corresponding plaintext and choose another message to send, adapting it to what he has previously seen. The aim is to quantify the $\mathcal{A}$'s likelihood in distinguishing between

encryptions of two messages, given that $\mathcal{A}$ can see encryptions of any message of its choice. The security game depicted below:

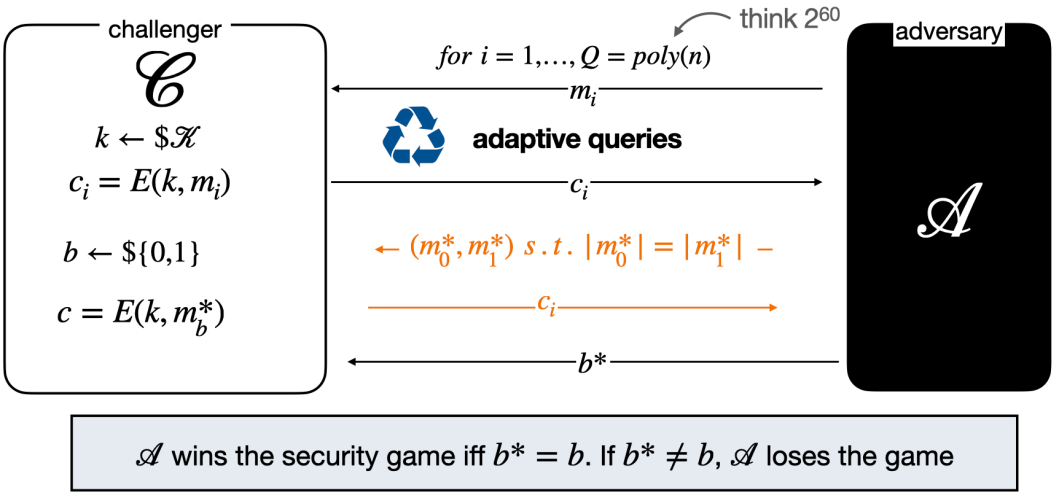


Figure 2: IND-CPA

We are giving the adversary $\mathcal{A}$ the possibility to learn many plaintext/ciphertext. The name "Chosen Plaintext Attack" comes from the fact that the adversary can choose the plaintext to be attacked on.

2.4 Advanced Encryption Standard

Data Encryption Standard (DES) is the most well-known symmetric-key block cipher. Introduced in the mid 1970s (refinement of IBM's Lucifer cipher) as the first commercial-grade modern algorithm with openly and fully specified implementation details. It is defined by the American standard FIPS 46–2. DES is now deprecated due to short keys.

In 1998, the National Institute of Standards and Technology (NIST) in the United States initiated a competition to develop a new block cipher that would one day supplant DES. The primary motivation for this effort was the relatively short key length of DES, which had become insufficient due to advancements in computing power—256 keys represented too small of a key space. A total of fifteen algorithms were submitted from around the globe. After a two rounds of peer review by cryptographic experts from all around the world, the selection was narrowed down to five candidates, which were further heavily scrutinized and cryptanalised. In the summer of 2001, NIST announced the chosen algorithm: Rijndael, which became known as the Advanced Encryption Standard (AES), featuring key sizes of 128 or 256 bits.

There are also other modern secure symmetric ciphers, used more rarely than AES: Serpert, Twofish, Camellia, RC5.

```
function AESK(M)
(K_0,...,K_10) ← expand(K)
s ← M⊕K_0

  for r = 1 to 10 do
     s ← substitute-nibbles(s)
     s ← shift-rows(s)
  if (r≤9) then
    s ← mix-cols(s)
  fi
  s ← s ⊕ K_r
  endfor

return s
```

Figure 3: How does AES works

As we see in the Figure 3, the idea is to expand the key, then essentially do OTP for 10 times loops. The nice

thing about the NIST competition is that everybody around the world was looking at the submission, and since 2001, so many cryptographers have checked the security of AES.

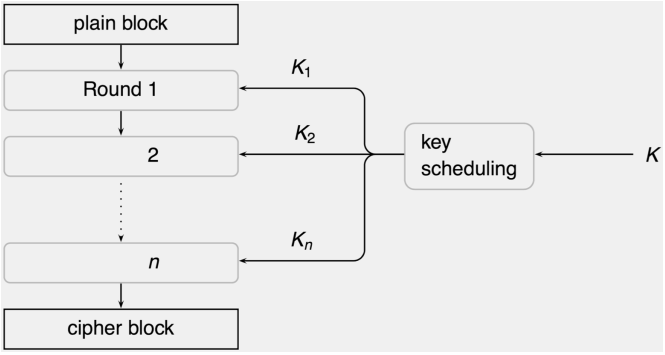
If you would like to have an intuition of how does AES work, see [1].

The idea behind all this is: if you cannot be a truly random permutation, at least strive to look like one. Idea: if we cannot be everywhere in the space, at least, we could be in a well-hidden portion.

2.5 Design Principles

The design principles for block ciphers are

Iteration : repeatedly apply a not-so-strong block cipher, each time with a different key (rounds).

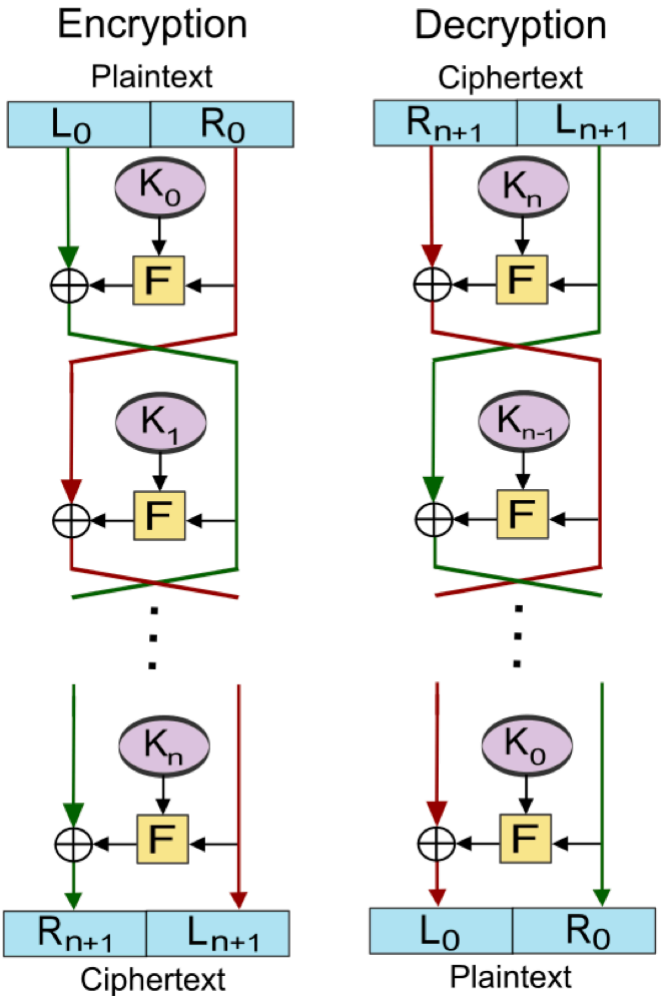


This technique is not provably secure, but heuristics show that it works!

Feistel Network : we take the input, we split it to left and right, and we pass it into a function F, then xoring, shifting and so on. The cipher function F is the same for every round. F does not need to be invertible for the round to be invertible and decryption is equal to encryption with round-keys in reverse order. The following relation between different round holds

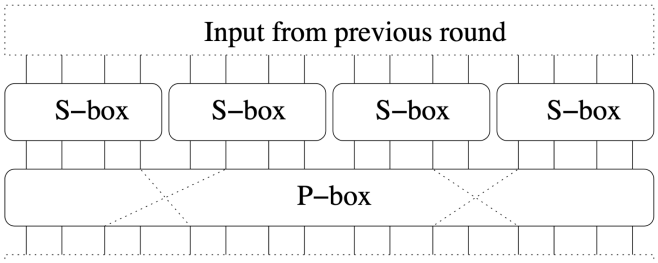
$$\begin{cases} L_{i+1} &= R_i \\ R_{i+1} &= L_i \oplus F(K_i, R_i) \end{cases}$$

[1] https://www.youtube.com/watch?v=gP4PqVGudtg



Confusion (S-boxes) : Divide the n bits of input into b sub-blocks of $\frac{n}{b}$ bits. Within each sub-block, apply a substitution table (**S-box**), i.e., a permutation on $\{0,1\}^{\frac{n}{b}}$, typically implemented as a lookup table. This introduces **confusion** to the cipher.

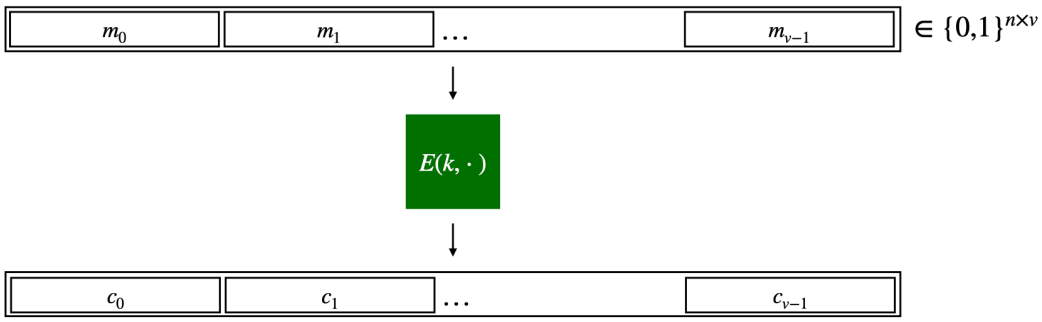
Diffusion (P-boxes) : Confusion is local; to spread its effect apply a transposition **P-box**, permuting bits between sub-blocks. This introduces **diffusion**.



3 Modes of Operation

A Block Cipher is a **deterministic**, **keyed** function that is **invertible**:

$$E(k,\cdot) : \{0,1\}^n \to \{0,1\}^n$$

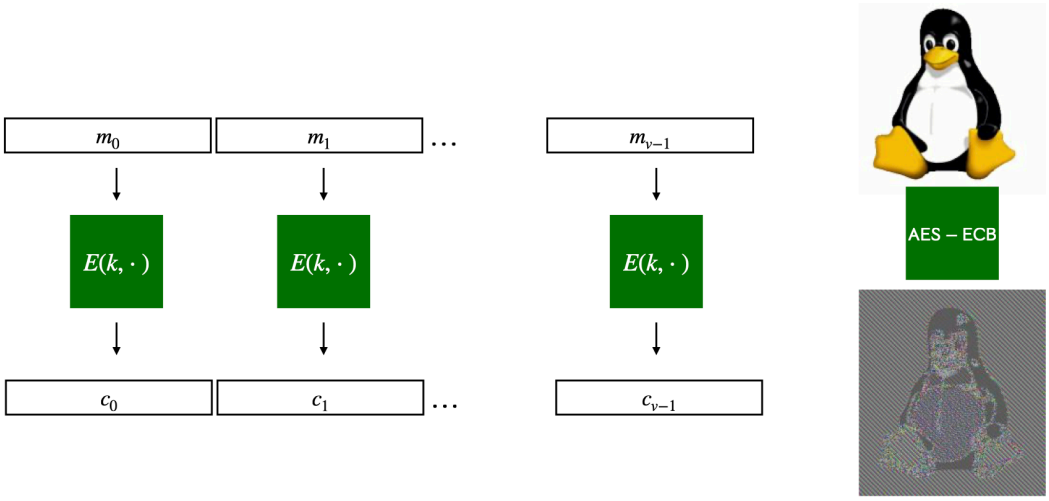


Let's look at a few options on how to operate over multiple blocks in a secure way.

3.1 Electronic Code Block mode (ECB)

The ECB can be described by the following mathematical formulas

$$c_i = E(k, m_i), \qquad m_i = D(k, c_i), \qquad \text{for } i = 0, \ldots, v-1$$



It is very easy to understand and implement and encryption and decryption are parallelizable (important for large data). The problem is that it lacks diffusion, it encrypts identical plaintext blocks into identical ciphertext blocks, nothing of one block infects other blocks. For this reason ECB is not recommended for use in cryptographic protocols. For example, in the penguin picture above, all the white blocks are encrypted in the same way, we can still see a pattern.

On the (in)security of ECB mode for multi-block messages AES is the current advanced encryption standard, so we can (safely) assume it is good block cipher. We have not delved too much on what it means for a block cipher to be secure, but you can live with the intuition that this means that the block cipher is a PRP, i.e., it is infeasible for an efficient adversary to distinguish ciphertexts from completely random blocks of the same length. This however holds for a single message, single block. When we consider long messages made of multiple blocks, the way blocks are encrypted may (and does!) affect security.

Consider a secure block cipher, e.g., AES, in ECB mode. We will show that this yields an encryption scheme that is **not** semantically secure. To do so, we need to provide an algorithm for $\mathcal{A}$ in the semantic security game that always guesses the correct bit with probability 1. Let n denote the block length, then:

- $\mathcal{A}$ chooses two distinct n-bit blocks, e.g., $block0, block1 \xleftarrow{\$} \{0,1\}^n$ with $block0 \neq block1$.
- $\mathcal{A}$ sets $M_0 = (block0, block1)$ and $M_1 = (block1, block1)$, and sends M_0, M_1 to the challenger.
- Upon receiving the challenge ciphertext $C = (c_0, c_1) \in \{0,1\}^n \times \{0,1\}^n$, $\mathcal{A}$ checks whether $c_0 = c_1$, in which case it returns $b^* = 1$; else $(c_0 \neq c_1)$, it returns $b^* = 0$.

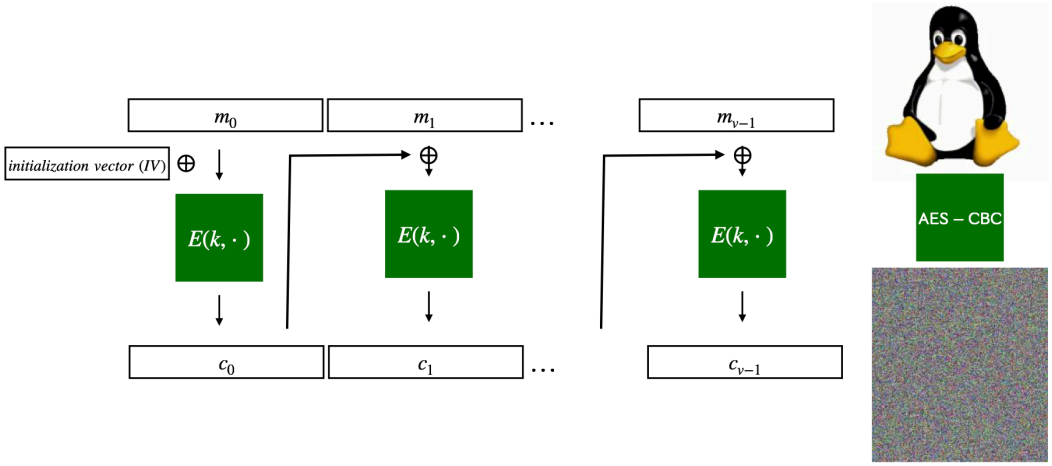
ECB is deterministic and has no "diffusion" across blocks, indeed it always encrypts the same input block into the same ciphertext block. This means that if the two blocks in the challenge ciphertexts are equal, they must have

come from M_1 (which is made of two identical blocks). Hence with this strategy $\mathcal{A}$ can win the semantic security game with probability 1. We can conclude that AES-ECB is not semantically secure (but luckily other modes of operations are secure). Note that this result implies that AES-ECB is not IND-CPA secure either (convince yourself that the very same attack would work in the IND-CPA game, if $\mathcal{A}$ makes 0 encryption queries).

3.2 Cipher Block Chaining mode (CBC)

Ehrsam, Meyer, Smith and Tuchman invented the cipher block chaining (CBC) mode of operation in 1972, the name comes from the fact that we chain blocks together. CBC is used in older versions of the TLS protocol (e.g., TLS 1.0). It is inferior to counter mode encryption. The CBC can be described by the following mathematical formulas

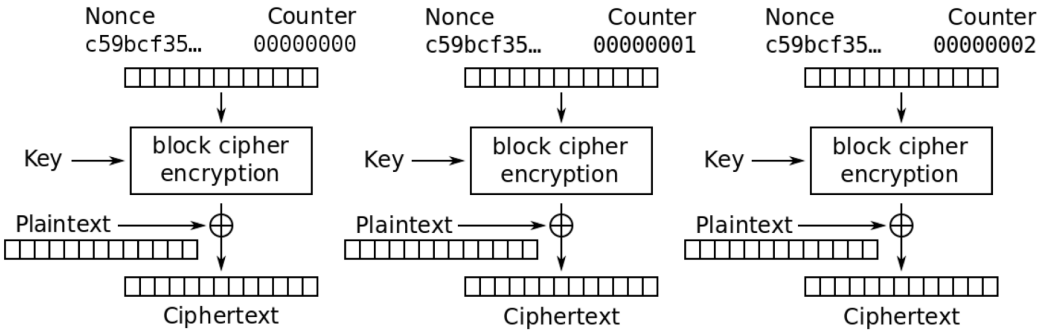
$$\begin{cases} c_0 &= E(k, m_0 \oplus IV) \\ c_i &= E(k, m_i \oplus c_{i-1}) \qquad \text{for } i > 0 \end{cases} \qquad \begin{cases} m_0 &= D(k, c_0) \oplus IV \\ m_i &= D(k, c_i) \oplus c_{i-1} \qquad \text{for } i > 0 \end{cases}$$



where IV is the *initialization vector*: it is the source of randomness and needs to travel with the ciphertext to enable the decryption of c_0.
One advantage is that decryption is parallelizable, but encryption is sequential (not parallelizable) CBC is rarely used because there are better options.
From the penguin picture above this scheme seems "secure".

3.3 Counter mode (CTR)

CTR mode was introduced by Whitfield Diffie and Martin Hellman in 1979. This is essentially a stream cipher. We input randomness in the block cipher, the nonce (a random number) and a counter. The name comes from the fact that a counter is involved in the scheme.

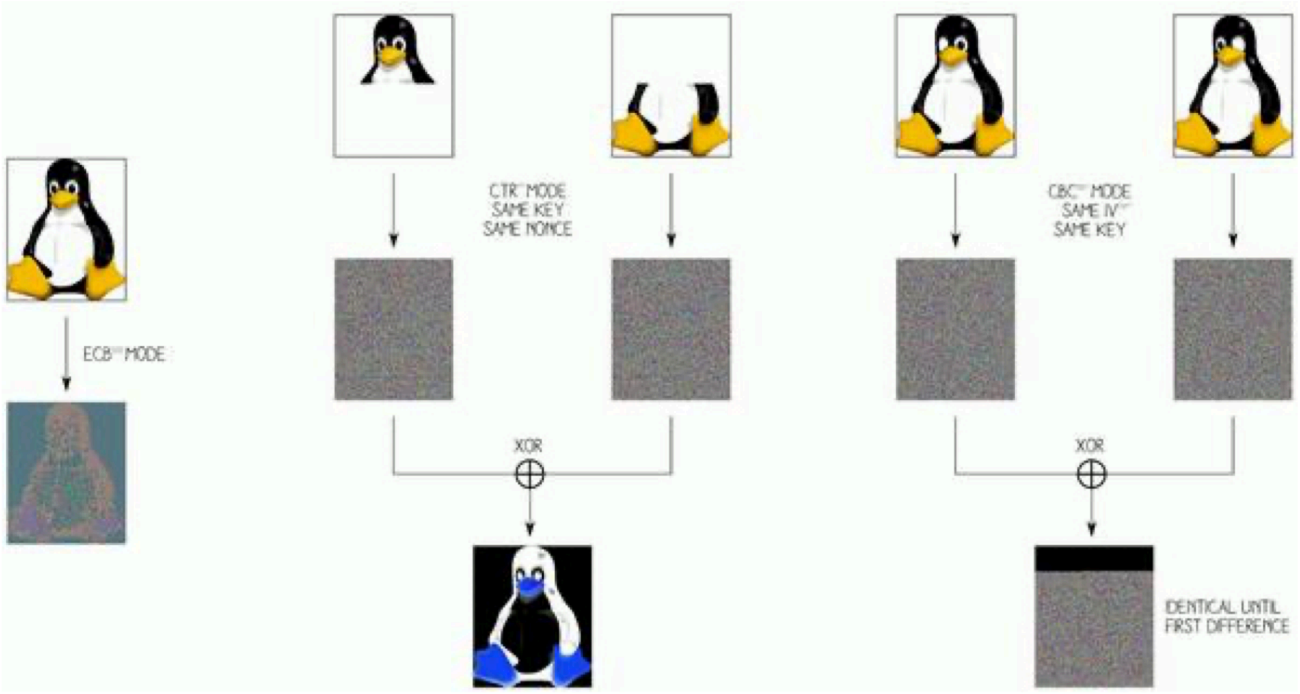


Counter (CTR) mode encryption

Encryption and Decryption are parallelizable (and basically E=D) but a drawback is that the nonce value is needed to decrypt every ciphertext block. CTR is used for cryptographic application but "with care": *nonce* means *number used once* otherwise you have problems.

3.4 Modes of operation's failures - visual examples

This picture summarizes some of the problems:



- **ECB** in the first picture we can see a pattern in the ciphertext;
- **CTR** If we split the penguin picture into two halves, we have the penguin back if we use the same nonce. The intuition behind that is that it is like if we are going to use the OTP with the same key twice.
- **CBC** They look completely different on their own, but when you xor them, they look different except for the first part, the first difference in the plaintext propagate, there is a diffusion principle.

TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Blockchain Technologies, Hash Functions

Lecturer: Victor Morel

Scribes: Lucia Lavagnino

Lecture 3 on 12 Nov

Last update November 27, 2024

Contents

1 Blockchains

Let us consider a historical perspective. In the pre-historical era, beads and shells constituted the first medium of good exchanges. With time, the first currencies were introduced (made in metals such as gold or silver) as a generic good for exchanges. The coins had a considerable intrinsic value –unlike today[1]– 1 euro is worth 0.0461 SEK of metal, while 1 Roman golden coin is worth more than 300 USD/3,328.75 SEK

But coins were (still are?!) heavy to carry around, can be easily stolen, and once gone, they are most likely lost forever. So we partially shifted to banknotes and other bills in the Middle Age. The value of paper currency depends entirely on its recognition and authenticity.

Another watershed moment in the history of currency was 1958 when Bankamericard (now Visa) was launched as the first credit card. Drastic change because means of payment became digitalized. Interestingly, credit cards only used encryption from 1996 with the introduction of Europay-MasterCard-Visa chips.

There is another medium of exchange: bitcoin, the latest contender as currency. Why is this trend, in your opinion? Answer: because of trust issues.

1.1 Trust

In the classic situation, pre-bitcoin, we have a bank centralized & fully trusted. We trust it fully, and the bank as an institution is centralized. It can know all your transactions, but privacy is limited.

But what if we do not trust the bank? For instance, you think the transaction fees are high, or because you lost some of your assets after a financial crisis. You could think of replacing the bank with a wide collection of anonymous and autonomous parties. Why? To avoid a central point of failure. However, if the entire system is distributed, we need a way to keep track of money and make sure it does not disappear or get stolen.

In this lecture, we will see what such anonymous and autonomous parties would need to do, in order to properly replace a central bank. Clearly, they need to define and agree on what transactions are valid. They also need to agree on the history of transactions, so that the same digital money can be spent once, but not twice. The digital history of recorded transactions is what we call a ledger.

Definition 1. *A **blockchain** is a growing list of items, called blocks, that are linked together by a cryptographic mechanism that ensures past blocks cannot be altered. This structure securely realises an append-only public ledger that can be used to broadcast information.*

[1] Source: https://www.911metallurgist.com/coin-metallurgy/.

There are two essential ingredients in a blockchain: the *chaining* mechanism that links blocks, and the *creating* mechanism to generate new blocks. In Nakamoto's proposal, the chaining mechanism is achieved by including the hash of the previous block as part of the next block. The block creation mechanism is called mining and consists in finding (via brute force) a `nonce` (aka salt, or random string) that makes the hash of the block one is mining start with a specific sequence.

Since a blockchain is distributed, everyone has a copy of it and can verify its content and authenticity.
It is also immutable, so that a malicious actor cannot just "invent" money out of the blue; nobody can reinvent history and pretend to be very rich, for example.
The last ingredient is a consensus mechanism, to agree on the new modifications to the ledger. Blockchains can be private, but private blockchains lose almost all interest: private blockchains lack decentralization, transparency, and openness, which are the core features that make public blockchains innovative and trustless. Private blockchains often resemble traditional databases, offering few unique advantages while losing the appeal of blockchain technology. Since public blockchains are more interesting than private blockchains, in this lecture we only focus on the first one.

1.2 Bitcoin

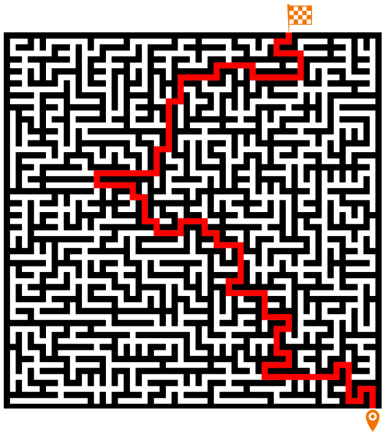
In 2008, Satoshi Nakamoto uploaded a white paper entitled Bitcoin: A Peer-to-Peer Electronic Cash System that proposes a solution to the long-standing problem of creating electronic cash without relying on any financial institution. This sparked the Bitcoin market and inspired a plethora of electronic payment systems. That are cryptographically secure and do not rely on a central bank.

In the paper, an electronic coin is defined as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

Because of trust purposes, all transactions are public, it means that everyone can scrutinize and call out fraudulent activities, which, in practice, means that no one lies. Before a set of transactions is added to the ledger (a block, as we shall see momentarily), a few conditions have to be met. The block needs to be verified, and the way to perform this verification step is by checking signatures in the block and make sure they verify, so that there are no fraudulent transaction is included in the block. Transactions in the Bitcoin network are secured using digital signatures. Cryptographic puzzles are solved by miners in order to add their block to the chain. The miner that solves the puzzle first gets a reward (now around 6.25 BTC equivalent to 4 512 070 SEK). This reward provides an incentive to perform this verification step was performed correctly. Transaction fees are also important because, at a certain point, if the blockchain continues for long enough, we do not have any blocks to mine.

1.3 Cryptographic Puzzles

To try to visualize these cryptographic puzzles, you can imagine a labyrinth with an entrance and one exit. This sets a well-defined problem: we know where to start, and where we should exit, but we need to figure out *how* to get to exist (the red path in the image to the right). Finding the path (the solution to the puzzle) is hard: you might have to try many possibilities, but once you have the solution, it's easy and fast to verify that it works! In summary, we can think of a cryptographic puzzle as a



- well-defined problem;
- hard to find a solution;
- but easy to verify it.

Two types of cryptographic puzzles are in bitcoin: one to transfer coins (digital signatures), and one to mine new coins. In module 2, we will talk more about digital signatures (and algebraic trapdoors); for this lecture we focus on cryptographic puzzles to mine new coins. There puzzles are defined by what is called a hash function. Before jumping to definitions, let us build an intuition for this cryptographic tool.

Think of a smoothie: given the smoothie, it is hard to find the exact ingredients and proportions, however, if you have the ingredients and the proportions, making the smoothie amounts to pressing the "on" button!
A slightly more formal but partial reformulation can be: given a binary string y of length d, find a value x such that a function $H(x)$ equals y. H is a cryptographic hash function, i.e., it is designed to be easy to compute, but computationally hard to "invert" (finding a pre-image of a given value y). We call one-wayness the situation in which a function is hard in one direction, easy in the other.

The cryptographic puzzle used to mine bitcoin is called proof of work and is connected to hash functions (and the particular hash function used in bitcoin is sha256).

Definition 2. *The **Bitcoin Hash Puzzle (Proof of Work PoW)** is build in this way*

- *Let* $y = (y_L, y_R)$ *and* $x = (x_L, x_R)$
- *Given* $y_L = 0^{19}$ *and* x_L*, find* x_R *s.t.* $sha256(x_L, x_R) = (y_L, y_R)$
- *Find a nonce value that makes a given input hash into a hex string with 19 leading zeros.*

You have two binary strings, and you want the output string to start with a specific sequence, (here nineteen leading zeros). The idea is to have one fixed part of the input x_L contain the block information, and try several random nonces x_R until you find one that satisfies this condition. But how do we adjust the hardness? Adding more zeros.
In bitcoin, the hardness is adjusted so that a puzzle is found every 10 minutes on average.

Example 1. *Let be* $str1 = IloveCrypto!$ *and* $str2 = IloveCrypto!-251509386766$*, then*

$$\begin{aligned} \mathsf{sha256}(str1) &= e8f6178df67ea4ec791b9fd72a2d710a3d832c113ee933a0654ae0e423d49ac9 \\ \mathsf{sha256}(str2) &= 0000092273023b5bc71c29852a01d0121336c16e700535cca2a8c5ef1459becd \end{aligned}$$

in this example, -251509386766 *is our random nonce.*

1.4 Immutable ledger

Now that we have seen how the Bitcoin hash puzzle works, let us move the focus to how to implement a ledger. The basic idea behind this ledger is to connect, or to link, blocks of data. A block partitions time into epochs/periods/time windows and it records anything that happens during that one time period. The idea is: since it is chained, the change in one block should affect all the other following blocks (here the name **blockchain**).

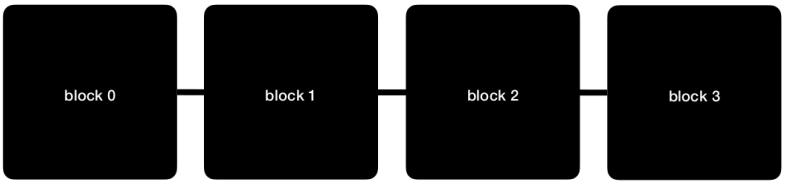
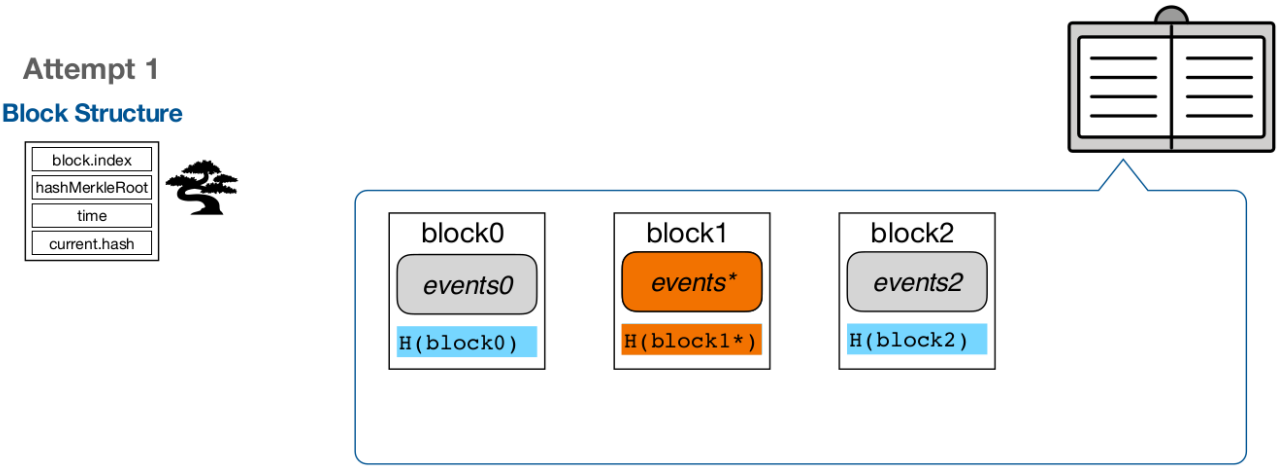


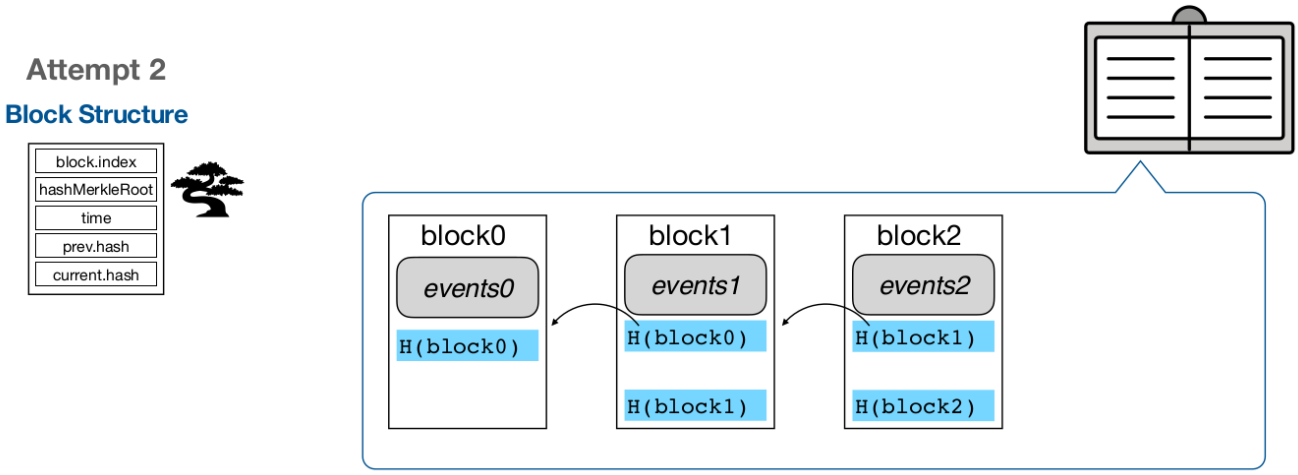
Figure 1: Example of a Block Chain, if Block1 is modify, also Block2 and Block3 change

In the blockchain contest, a fork occurs when a blockchain splits into two competing paths. The cause of forks can vary, for example, whenever a community makes a change to the blockchain's protocol, or basic set of rules. When this happens, the chain splits into two competing paths, producing a second blockchain that shares all of its history with the original, but is headed off in a new direction. There are also malicious fork acts, for example, trying to try to Double-Spending the bitcoin but the cryptographic nature of PoW ensures that adding new blocks to the blockchain is computationally difficult.

We can try with a **first attempt** at building such a ledger. We have to define a block structure, with an index, a data structure called a Merkle tree, of which we keep the root, it's used to efficiently store lots of data, in our case the transactions. We also store the time, and the hash of the block to be able to quickly verify it. Although it is working, there are a series of problems with this simple approach. We only have the blocks, not the chain! This means that there is no proof of work (anyone can mine a block) and no immutable history of transactions.

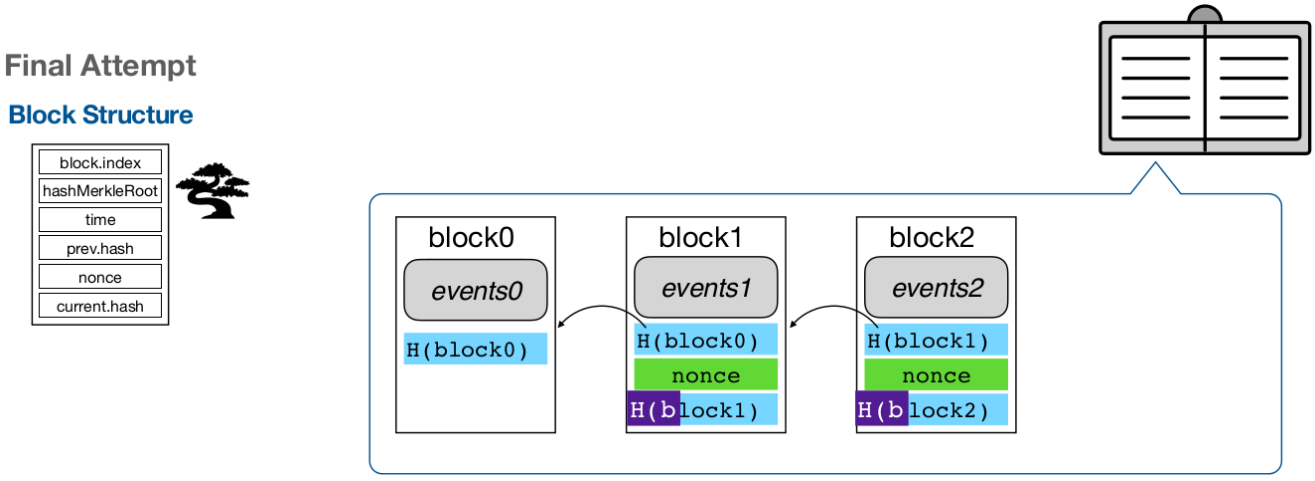


In our **second attempt**, we connect the blocks by adding the hash of the previous block.



This is better because we have the chain, but is the past really immutable? Not really, the hash of a tampered block can be computed efficiently, hence anyone can (easily) mine a block.

Final attempt, this time we add the proof of work with nonce: we need to find a nonce value that makes the hash of the current block start with 19 leading zeros . Anyone who wants to mine a block needs to solve the challenge and start with the right number of zeros, this takes a long time to practice. For a malicious actor to alter history, they would have to solve a proof of work for all the following blocks before the transaction is accepted. Today, bitcoin waits for several blocks (6 on average) before validating a transaction, hence it's unfeasible.



We now consider the last properties of a blockchain, the consensus mechanism. We need rules for the ledger:

- the past is immutable, we do not rewrite or alter history;
- everyone agrees on the history, through a consensus mechanism;

- we always build on the longest branch if there is a fork (longest chain rule);
- each block hash needs to start with a given prefix (lower the chance that blocks appear at the same time, uses the proof of work)

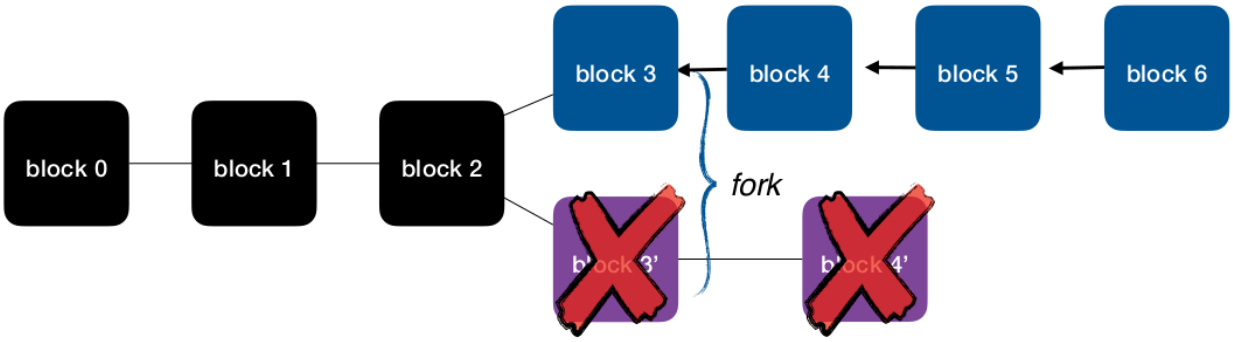
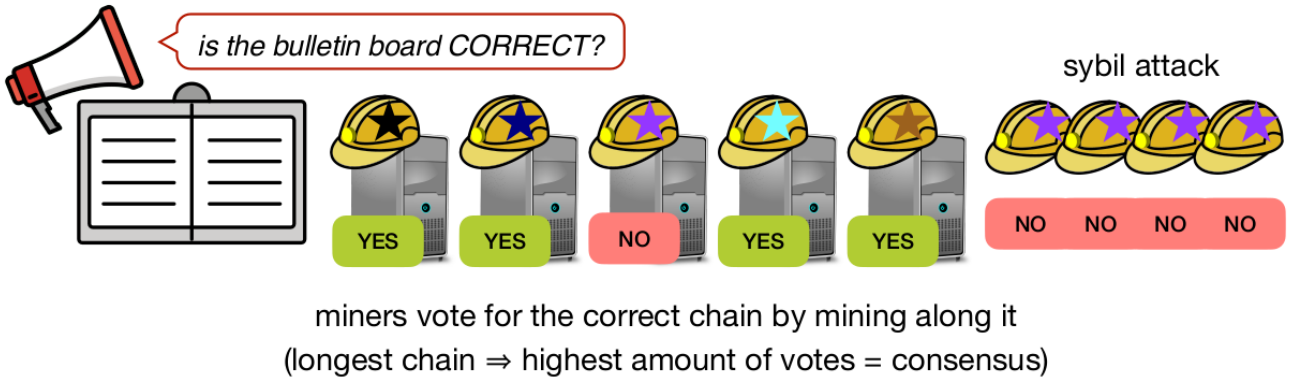


Figure 2: Because of longest chain rule, we are going to build on the blue chain (the longest)

If multiple branches live, then you can spend your money twice! (double-spending). A thing that can happen in theory is the appearance of two blocks at the same time, but because of the hardness of the proof of work, it is not happening in practice.

1.5 Consensus & attacks

To describe in more details the consensus mechanism, let us explore the longest chain rule mentioned before. The idea is simple, it is a voting process, for which mining a branch amounts to voting. If you want to vote for the correct chain, you mine it, and as such, it grows longer than the other branches. In this way, we have a consensus mechanism!
What happens if you have an attacker trying to perform what we call a Sybil attack? A Sybil attack is when the attacker creates multiple identities to gain disproportionate influence. The name comes after a book in which a woman called Sibyl had several personalities.
How does Bitcoin prevent such attacks? Answer: thanks to the PoW, because having more multiple identities does not create additional computational power. Hence, it is prohibitively costly to steal a consensus.



It is costly to steal the consensus in big blockchains such as bitcoin: in bitcoin, there are too many people who require thousands of data centers, hence really high computational power.
The problem is that it is feasible to steal the consensus in small blockchains and then vote for a forgery branch.

In this paper from 2020, these types of forgery that allowed to double-spent the money happened: in the paper has been denoted the 51 percent attack. Luckily, it happens mostly on small blockchains, where the amount of computational power needed to topple the voting process is more reasonable. The importance of having a robust consensus mechanism is demonstrated in scenarios where money can be stolen. If you can steal, the consensus, you can do what is called double-spend digital cash.

1.6 Proof of work vs. Proof of stake

It is important to point out that bitcoin, because of the proof of work, is terrible for the environment (if you are interested, see this news article). For cryptographic hash puzzles, the hardness is adjusted so that a new block is added every 10 minutes on average. The hardness is adjusted because miners are getting more and more computational power, which means that more and more power is spent on mining.

That has repercussions on the environment in numerous ways:

- more GPU are built for that (requiring rare metals);
- more energy is required to run datacenters mining bitcoins':
- and more water is needed to cool off machines.

However, and although the initial design of the bitcoin blockchain uses the costly proof of work, other mechanisms have been proposed in the academic literature. There is another consensus mechanism called **proof of stake**. This mechanism doesn't rely on computational power and cryptographic puzzles, but takes an entirely different approach.

The core idea is to change the rule for selecting the lottery winner, the validator that will be rewarded with new coins. Here we replace work with stake, or computational power with digital cash. The probability of being selected (so mining the new block) is proportional to your bid: the higher you bid and the more likely you are to win.

The proof of stake is currently in use in several blockchains, probably the most famous of which is Ethereum. Ethereum actually transitioned from PoW to PoS for its $v2.0$, demonstrating that it is possible for a big blockchain to make such a big change.

We do not have any miners anymore, basically, we lock the next block temporarily, everyone bids, and the winner gets to mint the new block. We want to add the new block to the blockchain, in the block, there are all the transactions with all the transaction's data. Mint is not a typo for mine, it is how you describe the creation process of new money in economic terms. The winner gets to check all transactions in the block, and gets the reward. Here the incentives are the transaction fees and the block reward off, but it is complemented with the fact that the validator can lose part of their stake if they approve fraudulent transactions. The miners in proof of stake became validators. This last component in this consensus mechanism drives participants towards good behavior.

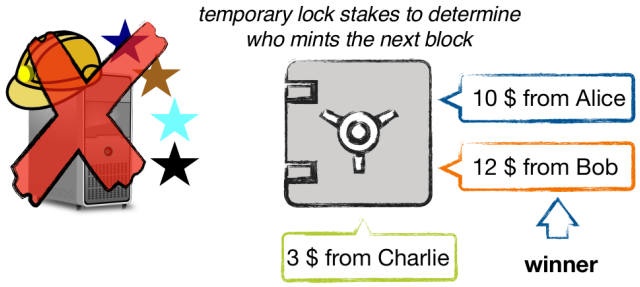
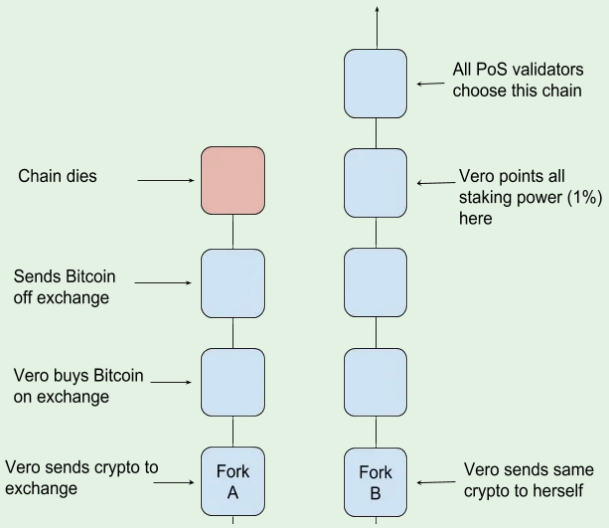


Figure 3: The probability of being selected is proportional to your bid: Bob wins since he has the highest probability to be selected, although having the highest probability does not ensure a win!

There is an attack against proof of stake, denoted the **nothing at stake theory**. This attack also enable double-spending, but it requires specific preconditions:

1. validators always prioritize profit, this is a standard security assumption in the crypto world;
2. no one altruistic, no validators will mine on only one chain at a time, all people try to maximize their benefit;
3. a fork has occurred and validators are actively building both forks;
4. and software is modified to do mine on all forks.

Example 2. *First, let's assume that a fork has occurred and all validators are actively building on both forks, As we see in the figure below.*
Veronica, our attacker, is interested in executing a double spend attack. She sends her crypto to an exchange in one fork (fork A) and to a wallet she controls in the other fork (Fork B). After enough time passes, the exchange accepts her deposit because it is recognizing fork A. Veronica then buys some Bitcoin with her deposit and quickly takes that Bitcoin off the exchange.
Now she points her 1% staking power to fork B. Eventually she is selected to validate a block and builds only on Fork B. Being that everyone was building on both and there wasn't a clear "longest chain" , the entire network now converges on (aka chooses) Fork B.
Veronica has now successfully stolen Bitcoin off an exchange. The crypto she sent to the exchange has returned to a wallet in her control and now she has some extra Bitcoin too.



1.7 About some assumptions

How anonymous is Bitcoin? Bitcoin is weakly anonymous. Parties are anonymous in the sense that only public keys are revealed, people do not need to declare their real identities. One's real identity is therefore hidden in theory, but the transaction graph is public, it can be used to deanonymize.
However, there are other blockchains, different from bitcoins, in which anonymous transactions can be performed. Zcash provides the possibility to perform *anonymous* (shielded) transactions using Zero-Knowledge proofs. Note that many exchanges in practice require a proof of identity to open a wallet, in order to fight fraud, hence the protocol in theory is providing anonymity, but the implementation is not.

Are cryptocurrencies entirely solving trust? We said that the problem that cryptocurrencies are addressing is trust, and that is true, but we would like to challenge the fact that they are actually solving this problem entirely. Indeed, blockchains work because we trust the Mathematics behind the crypto: the PoW is infeasible to break. What they do is operate a shift in the trust from institution to technology. However, it can come with some downsides because, as the math is adamant, and their guarantees have been formally proven, blockchains are also complex technological artifacts. Their code can have bugs (and people lost money because of that), and wallets get stolen without recourse.

While you can go to your bank if you think someone has been stealing from you, and get a refund, it is not the case for crypto wallets. Others lost their keys to their wallets (amounting to 1.8 million BTC, or 121 billion USD), and some exchange places froze wallets too based on suspicion of fraudulent activities. Finally, you still need to trust people to a certain extent, to change the block size, for instance. Typically, we saw that this shift of trust was largely due to a shift in the trust distribution, from a centralized entity to a distributed set of parties. And indeed, this protocol is distributed, but the technology stack is neither few exchange places nor one main company for the mining hardware.

Are cryptocurrencies even currencies in the first place? To pursue on the critique of concepts, we would like to challenge the notion that cryptocurrencies are currencies in the first place. As a matter of fact, economists commonly agree on 3 criteria to define a currency: in economy, a **currency** is:

- a unit of account;
- a store of value;

- a medium of exchange.

The problem with seeing Bitcoin as a currency is that it fluctuates too much, and additionally it does not have intrinsic value (indeed bitcoin is entirely digitalized and intangible). Although Bitcoin could be consider as a medium of exchange, transactions fees are proportionally high for small operations, and no refund possible.

To give a real world example, El Salvador is the only country in the world has legislated on bitcoin as a legal tender, and it's not their only legal tender. So if we want to be accurate on the terms, we should call them crypto-assets instead of crypto-currencies.

2 Hash Functions

We are going to study more specifically the mathematical definitions of hash functions. As a reminder, hash functions form the basis of cryptographic hash puzzles used in the proof of work, they are easy to compute and hard to invert, here the easy and hard are to be understood according to computational complexity, whether they can be solved in polynomial time. We can use the smoothie as a metaphor to understand the hash function: is both "easy" to compute (easy to blend) and "hard" to invert (hard to figure out the ingredients and proportions).

2.1 One-way functions

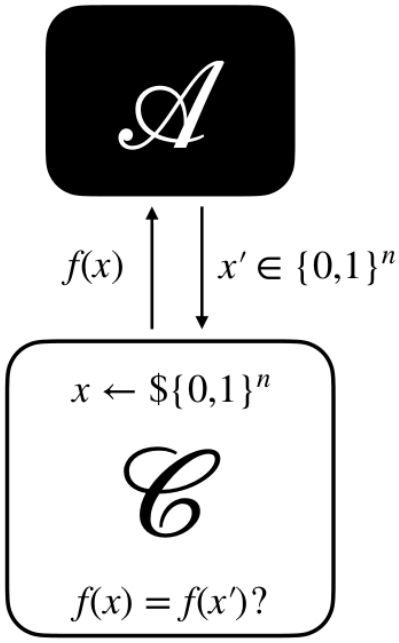
Hash functions are a generic term that denotes a function that maps an input to an output usually of fixed-length. A specific implementation, useful in our case, are one-way functions.

Definition 3. *A function* $f : \{0,1\}^n \to \{0,1\}^d$ *is* ***one-way*** *if:*

- $f(\cdot)$ *is efficiently computable, i.e. there exists an algorithm that computes* $f(x)$ *in polynomial time for all inputs* $x \in \{0,1\}^n$
- *for every PPT algorithm* $\mathcal{A}$ *there is a negligible function* $negl_{\mathcal{A}}(\cdot)$ *such that for sufficiently large values of* $n \in \mathbb{N}$ *it holds that*

$$Pr[f(x) = f(x') | x \xleftarrow{\$} \{0,1\}^n, x' \leftarrow \mathcal{A}(f(x))] \leq negl_{\mathcal{A}}(n)$$

In the definition, the second condition is the mathematical term for *'the function needs to be hard'*. The formula says: if we take a random input x in the domain of f, we give to the adversary $f(x)$ and the adversary gives us back x', then the probability that $f(x)$ is equal to $f(x')$ should be less or equal to a negligible function. This property is equivalent to the security game drawn to the right (and stanting the the probability that $\mathcal{A}$ wins the game should be negligible).
In simple terms, a one way function is both: **easy to compute** and **hard to invert**.



How do we build such functions? An algebraic candidate is integer factorization.

Example 3. *OWF from **integer factorization**:*
consider $f : \{k$ - bit primes$\} \times \{k$ - bit primes$\} \to \mathbb{N}$, for $k = 256$, defined as: $f(p, q) = p \cdot q$.
$f(\cdot)$ is a one-way function if integer factorization is (computationally) hard. Here we take two primes of k-bit length each. Easy to compute, but if you have the product, it is very hard to invert and find the primes that computed it. What happens if we consider random primes? The answer to the question is that for unbounded length, some factorization is easy.

In 1996, Peter Shor constructed an algorithm that can factor integers easily, this was the first quantum algorithm.
A security note though, OWF only guarantees that the input x is not leaked entirely, which in practice means that it is still possible that $f(x)$ leaks a substantial amount of information about x. In the following example, we are going to show an example of one-way function that leaks half of the input.

Example 4. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a OWF.*
Consider the function $g : \{0,1\}^{2n} \to \{0,1\}^{2n}$ defined as $g(x_0||x_1) := f(x_0)||x_1$.
Even if $g(\cdot)$ reveals half of its input, it is still a OWF!
This is because $f(x_0)$ is still a OWF anyway, so $g(\cdot)$ not being an OWF implies by contradiction that $f(\cdot)$ is not an OWF.
In details: since "$f(\cdot)$ is a OWF $\Rightarrow$ $g(\cdot)$ is a OWF" is logically equivalent to "$g(\cdot)$ is NOT a OWF $\Rightarrow$ $f(\cdot)$ is NOT a OWF", in order to prove that "$f(\cdot)$ is a OWF $\Rightarrow$ $g(\cdot)$ is a OWF" we can prove that "$g(\cdot)$ is NOT a OWF $\Rightarrow$ $f(\cdot)$ is NOT a OWF" .
If $g(\cdot)$ is NOT a OWF, it means that: there exists a PPT $\mathcal{A}$, to which, given $g(x_0||x_1)$ can infer $(x'_0||x'_1)$, such that

$$\begin{aligned} g(x_0||x_1) &= g(x'_0||x'_1) \\ &\Downarrow \text{ for the way in which } g \text{ is defined} \\ f(x_0)||x_1 &= f(x'_0)||x'_1 \\ &\Downarrow \text{ removing } x_1 \text{ and } x'_1 \text{ from each side, comparing the first half of the string} \\ f(x_0) &= f(x'_0) \end{aligned}$$

that means in other words "f is NOT OWF" .
We demonstrated the equivalence of the proposition using the negation of the implication.

2.2 Collision Resistance

A special kind of OWF really important in cryptography are Hash Functions.

Definition 4. *A **Cryptographic Hash Function** $H : \{0,1\}^n \to \{0,1\}^d$ has the following properties:*

- *pre-image resistance(i.e one-way):*

$$Pr[H(x) = H(x')|x \xleftarrow{\$} \{0,1\}^n, x' \leftarrow \mathcal{A}(H(x))] \leq negl_{\mathcal{A}}(n)$$

- *compressing (i.e., $n > d$) and of fixed output length d;*
- *second pre-image resistance (i.e. collision resistant)*

$$Pr[H(x) = H(x')|x, x' \leftarrow \mathcal{A}(H), x \neq x'] \leq negl(n)$$

In a nutshell, Hahs functions are OWF with extra properties:

- like OWF they are one-way, this property is called **pre-image resistance** because it means that you cannot easily find a pre-image. Facing a PPT adversary $\mathcal{A}$, the probability that an adversary outputs x', given $H(x)$, such that the output of the two hashes x and x' are equivalent is lesser than a negligible function.
- Another property is that it is **compressing**, the output domain must be smaller than the input domain, and we like to have a **fixed output length**, i.e. d is a constant value.
- Finally, the main addition is the collision resistance, also called **second pre-image resistance**. This property means, facing a PPT adversary $\mathcal{A}$, the probability that an adversary outputs x and x' such that the output of the two hashes on x and x' are equivalent is lesser than a negligible function.

Note that this definition does not mean that there exist **no** collision for hash functions, just that they are unlikely.

Cryptographic hash functions are widely used to store an obfuscated version of a piece of information in a deterministic way (e.g. passwords).

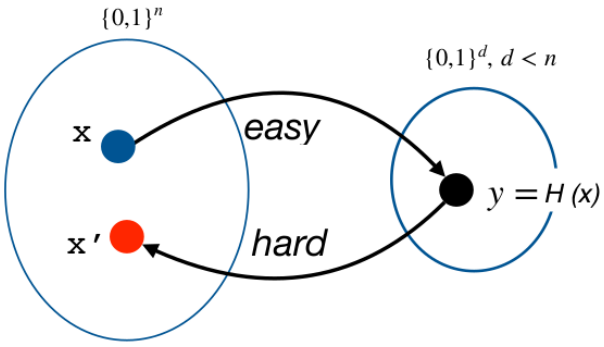


Figure 4: Definition of Hash Function

2.3 The Birthday Paradox

Can we estimate the probability of finding a collision?
As we saw, it is possible to find collisions if the pool of answers is low enough. But can we mathematically estimate the probability of finding a collision?
Here comes the so-called "*birthday paradox*" . In probability theory, the **birthday problem** asks for the probability that, in a set of n randomly chosen people, at least two will share a birthday. The **birthday paradox** refers to the counterintuitive fact that only 23 people are needed for that probability to exceed 50%.

The intuition behind the birthday paradox:

- there are 365 days a year, hence the probability that a given pair of birthdays are different, is $1 - \frac{1}{365} = \frac{364}{365}$;
- for n people, there are $\binom{n}{2} = \frac{n(n-1)}{2}$ possible pairs;
- the probability of having n different birthdays is like "throwing the birthday dice" as many times as you have pairs, or $\left(\frac{364}{365}\right)^{\frac{n(n-1)}{2}}$ **not** to have a collision (independent events);
- now we calculate the complement to have the probability of a collision, this will be $1 - \left(\frac{364}{365}\right)^{\frac{n(n-1)}{2}}$.

Example 5. *For 23 people in a room, if we take the number of possible combinations, we have* $\binom{23}{2} = \frac{23 \cdot 22}{2} = 253$ *possible pairs. Take the probability of two people having two different birthdays, which is* $\frac{364}{365}$*, flip that to the power of 253, and we have the probability of 23 people NOT having the same birthday, which happens to be very close to* 50%*. What happens if you have, say, 75 people in a room? Then we have* $1 - \left(\frac{364}{365}\right)^{\frac{75 \cdot 74}{2}}$*, so the probability is closed to* 99.95%*.*

Note: the probability if $n > 365$ is equal to 1 because of the Pigeonhole principle, if we have more than 365 people in a room, since there are *"only"* 365 days in a year, for sure, at least 2 of them will share the same birthday.

What about hash functions? Two persons having the same birthday is replaced with the hash function outputs. For hash functions:

- We consider a function $H(\cdot)$ acting as a purely random function (Random Oracle Model)
- "Days a year" for Hash functions is the size of the hash digest N; "people in a room" are the attempts to find a collision;
- with a large enough output domain, it is unfeasible in practice (see Sha256).

2.4 Merkle Trees

Merkle Trees are a cryptographic data structure used to efficiently store large amounts of information, and to efficiently verify this information. It's a binary tree (see Figure 5), which is to be understood as having max two branches, but trees can be unbalanced in practice.

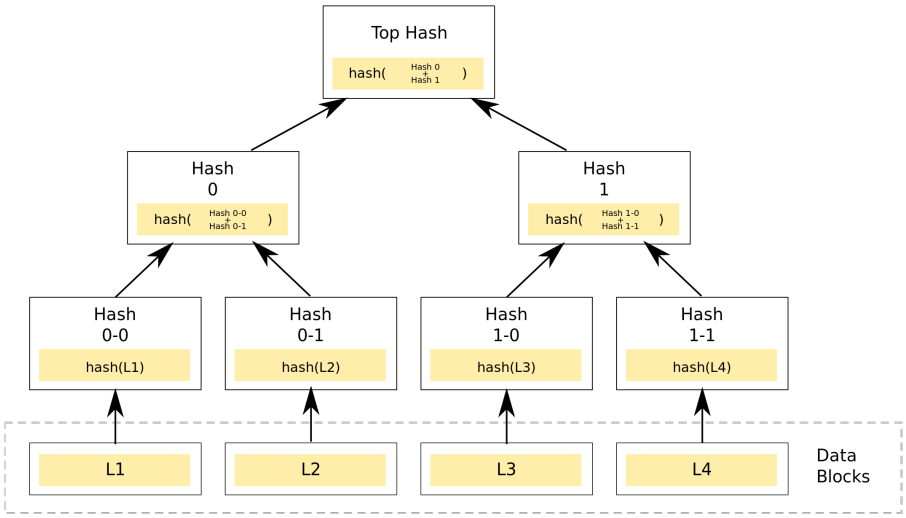
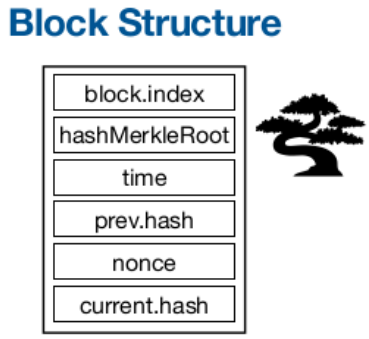


Figure 5: The leaves of the MT are the Data Blocks (the information we would like to store). The MT stores information in a very elegant way: first hash the leaves, concatenate and then hash again, until the root. If you want to verify the integrity of the MT and you have for example Hash1 and you know L1 and L2, you do not need to have access to the data L3 and L4. First hash L1 and L2, to obtain $\mathsf{Hash0}-0$ and $\mathsf{Hash0}-1$, concatenate them and hash to gain Hash0. Finally concatenate Hash0 and Hash1, and hash, to have the root of the tree. In this way, you can easily verify the integrity of the tree without knowing all the data.

We have our data blocks, as leaves, in the figure above represented at the bottom. We hash your leaves, and store the result in our parent node. To construct the parent node 0, we hash the concatenation of 00 and 01, etc until the top hash, more commonly known as the root of the tree. The representation can be a bit confusing, but trees, when we talk about data structures, are represented upside down. The main thing to remember about merkle trees is that the hash of the root uniquely identifies its content (considering that there is no collision), namely the **data blocks**.

Now let us see how they are used in the wild, starting with the running example of Bitcoin.

Merkle trees are used to store data about transactions. In a block, instead of storing all transaction data, only the root of the tree is stored and used to verify the content of the transactions.
Another interesting use of merkle trees is on the web, in a security standard called certificate transparency. When you register your website, if you provide an encrypted access to your website and want to prove that you are indeed the rightful owner of the public key used in the encryption scheme, you register a certificate signed by a trusted authority. To improve web security, certificate transparency (CT) makes public all certificates in a distributed ledger, that anyone can audit. And to improve the efficiently of the verification step, these certificates are stored in a merkle tree. More information at this link and in the intro of this PhD Thesis .



Merkle trees are also used in a very popular version-control software called git . Git hashes all the code and other executable files, and keeps the root to uniquely identify your commit.

Do we need a blockchain or a Merkle tree? Blockchains are needed in specific situations but the cost of implementating and maintaining them is rather prohibitive. Nowadays, there is a tendency to just apply blockchains to whatever problem, without considering whether it is the best tool available. Blockchains can be heavy, you have to set up a large infrastructure (which costs money, energy, and human resources), and as we have seen before, the proof of work costs a lot of energy but is still prevalent, and even if we see a transition to proof of stake, it is a slow transition. Moreover the prevalence of the Proof of Work has significant ecological impact.

To certify some documents such as diplomas, a blockchain is not required:

- diplomas are official documents certified by trusted third parties (e.g. Chalmers);
- there can be an interest in maintaining a distributed ledger of diplomas;
- in this context, not everyone should be able to write in the ledger;
- also, a consensus mechanism is not required;

therefore, a Merkle is sufficient. We are bringing up this example specifically because the French university of Lille recently used a blockchain for this purpose but they have been criticized for the unnecessary character of this implementation.

TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Message Authentication Codes, Authenticated Encryption

Lecturer: Elena Pagnin Lecture 4 on Nov 15

Scribes: Lucia Lavagnino Last update December 4, 2024

Contents

1 Secure Communication

So far, we have learned that any block cipher in ECB mode yields a deterministic encryption that is not semantically secure for long/multiple messages. Indeed, if the same key is used to encrypt the same block, the resulting ciphertext will always be the same. On the other hand, a block cipher in CBC and CTR mode uses randomness (IV and nonce, respectively), hence two encryptions of the same plaintext under the same key will (generally) produce two different ciphertexts.

The bottom line is that a good (secure) encryption scheme should be randomized. Hence, we want **randomized** (aka probabilistic) encryption. The problem is that probabilistic encryption generates ciphertexts larger than the plaintext. This is because it is necessary that the ciphertext provides information about the randomness, not just the message. This is an inevitable price to pay for better security, but in certain settings we can get good security without ciphertext expansion. We are going to see a new security notion that is stronger than the previous one, but still achievable by randomized encryption.

1.1 Towards IND-CPA Security

Now we introduce a fundamental security notion in the field of cryptography, specifically related to the security of encryption schemes. This notion provides a formal framework to evaluate how secure an encryption scheme is against adversaries that can adaptively choose plaintexts. In this game, the adversary's goal is not to decrypt a ciphertext (this would be too hard and we have seen that damage can be done with much less); rather, $\mathcal{A}$'s goal is to gain some information about the plaintext concealed in the ciphertext. In order to quantify this *leakage* of information, we introduce the notion of indistinguishability. This notion is weaker than perfect secrecy.

The adversary $\mathcal{A}$ would like to distinguish between the encryption of two known plaintext messages. In crypto jargon: **ind**istinguishability under **c**hosen **p**laintext **a**ttack (IND-CPA). IND-CPA has many equivalent notions, read here if interested. The term *chosen plaintext* comes from the fact that, in the security game, $\mathcal{A}$ chooses the (two) plaintexts it wants to be challenged on. The noun *indistinguishability* refers to the fact that if the adversary knows the plaintexts, it still cannot distinguish from which message the challenge ciphertext it receives was generated. When you see this, you can think that is "artificial": where on Earth do you have that the adversary can choose precisely what message the challenger is going to send? Instead, we have witnesses that this happened.

Historically, an example of this tactic can be seen in British military strategy, where mines were strategically placed in specific locations to prompt German forces to send encrypted messages referencing those areas. In a modern context, this can be likened to an attacker embedding malicious JavaScript in a webpage, which then compels a victim's web client to establish an HTTPS connection.
Hence, we are giving to the adversary $\mathcal{A}$ a really strong power, but it is not unrealistic. If we can prove security even in this bad scenario, we are secure even against weaker adversaries.

To better understand the adversary, we can summarize its goals and powers as follows:

Goal : to distinguish between the encryption of two known plaintext messages **IND-CPA**

Power : $\mathcal{A}$ should be an efficient algorithm (probabilistic, and that runs in polynomial time $< 2^{60}$); $\mathcal{A}$ can see everything transmitted over the communication channel; $\mathcal{A}$ knows all details of the encryption scheme except for the secret key, in accord with Kerckhoffs's principle.

Security Game (IND-CPA) The security game for indistinguishability under chosen plaintext attack (IND-CPA) is described by the following figure. The aim of this security game is to quantify the adversary's likelihood in distinguishing between encryptions of two messages, given that, $\mathcal{A}$ can see encryptions of any messages of its choice. If the adversary wins the IND-CPA game only with negligible probability, then the encryption scheme under consideration is said to be IND-CPA secure. Otherwise, if $\mathcal{A}$ has non-negligible probability of winning in the game, the scheme is deemed insecure.

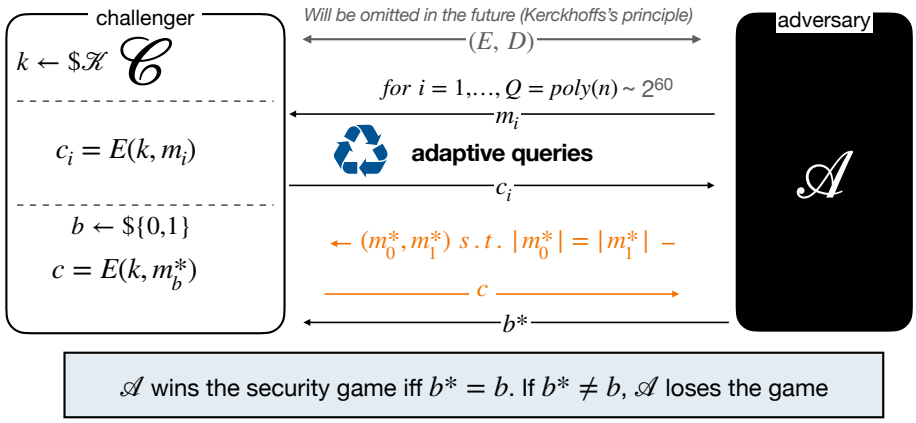


Figure 1: IND–CPA game

Verbose description of the IND-CPA security game In accord with Kerckhoffs' principle, the game begins by giving the description of the encryption and decryption algorithms to the challenger and the adversary. Since this step is common in all security notions, from now on we will omit to mention it. The security game runs the following steps:

1. (setup) The challenger $\mathcal{C}$ samples a random key $k \xleftarrow{\$} \mathcal{K}$;
2. (query phase) The adversary $\mathcal{A}$ sends at most Q queries consisting of a message m_i to $\mathcal{C}$. Upon receiving m_i, the challenger encrypts m_i using the key k sampled in the setup (the same key for all messages) and then returns the computed ciphertexts c_i back to $\mathcal{A}$. The queries are **adaptive** meaning that the adversary can ask for the first message m_1 and receives back the corresponding ciphertext c_1, then it can ask for message m_2, and so on, in a loop. The loop stops after at most Q rounds, where Q is a number polynomial in the security parameter.
3. (challenge phase) The adversary chooses two messages m_0^* and m_1^* of the same length, and sends them to $\mathcal{C}$. The challenger picks a random bit $b \xleftarrow{\$} \{0,1\}$ and encrypt the message m_b^*, obtaining the challenge ciphertext c. $\mathcal{C}$ returns c to $\mathcal{A}$.
4. The adversary returns its guess $b^* \in \{0,1\}$ to the challenger.
5. The challenger returns `win` if $b^* = b$, i.e., if $\mathcal{A}$ guessed correctly. Otherwise it returns `lose`.

About adaptive queries Adaptive queries are a powerful tool, since they allow the adversary to choose its next query based on the answers it received for its previous queries. In other words, adaptive queries are a run as a loop of polynomial length (think up to 2^{60} queries). For example, $\mathcal{A}$ can ask for $m_2 = 2 \cdot m_1$ and see what happens to the corresponding ciphertext. Depending on the encryption scheme, $\mathcal{A}$ may also use c_1 to generate the next query message. The challenger gives the adversary Q many queries, where Q is polynomial in the security parameter. Note also that queried messages can have different length. This is not the case for the challenge messages, otherwise $\mathcal{A}$ can trivially distinguish the encryption of a short message by a long message just by checking the length of the challenge ciphertext.

Definition 1. An encryption scheme is said to be **indistinguishable under chosen plaintext attack (IND-CPA secure)** if any PPT adversary $\mathcal{A}$ that engages in the IND-CPA game has only negligible advantage in winning:

$$Adv(\mathcal{A}) = \left| Pr[\mathcal{A} \text{ wins}] - \frac{1}{2} \right| < negl(n)$$

With the formula in Definition 1, we are wondering how far away are we from random guessing? If we are *negligibly close* to random guessing, then the encryption scheme is IND-CPA secure.

General Results Both AES-CBC and AES-CTR modes are IND-CPA Secure (the formal proofs are not part of the course).
In general, if (E, D) is a secure block cipher, then:

- using (E, D) in CBC mode yields an IND-CPA secure cipher;
- using (E, D) in CTR mode yields an IND-CPA secure cipher.

Padding When splitting a long message into blocks, we need to take care of padding, whenever the original message's length is not a multiple of the block length. Padding is often needed before encryption with a block cipher: a message must be as long as a integer number of full blocks in order to be encrypted with a block cipher. As an essential property, padding must be **reversible**, i.e., the receiver must be able to remove padding in a unique way. An example of insecure padding could be done by adding necessary number of zeros to fill last block. This is very bad for security because then the adversary $\mathcal{A}$ knows that the last part of the message is all zeros (on the web there are several witnesses of *bad padding* attacks). Plus it is also very bad for correctness: the receiver will never know where the original message ends, is it for SEK10 or SEK10000? The receiver should always check that the padding is correctly applied before removing it. In case of mismatch, the protocol should be immediately aborted. Note that the adversary $\mathcal{A}$ always knows the padding using in the scheme (by Kerckhoffs's principle).

Confidentiality VS Authenticity So far we have discussed a secure communication over an insecure channel, where the adversary $\mathcal{A}$ is listening all the conversation. Until now we have been concerned about confidentiality (keep the content of the conversation unintelligible to external parties). What happens if the adversary $\mathcal{A}$ has the power to change messages? The next goal is to work with **Integrity/Authenticity**: anybody should not be able to modify messages in an undetectable way, or to impersonate a sender.

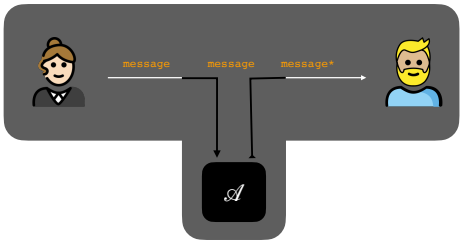


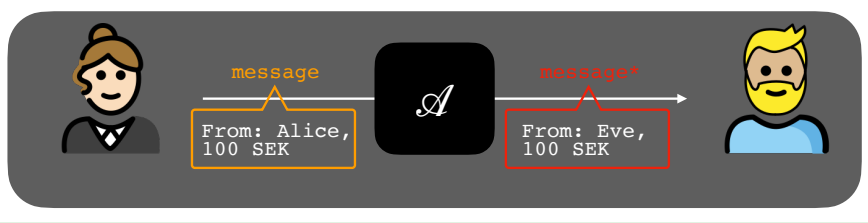
Figure 2: Example of message manipulation, the received $message^*$ may be different from the sent $message$.

- integrity means that we want to be sure that the data that you received is intact and is the same authentic data sent;
- authenticity means that we have a proof by the sender that what we have is the original message sent; we would like to be able to detect if there is some modification.

1.2 Why Does Integrity Matter?

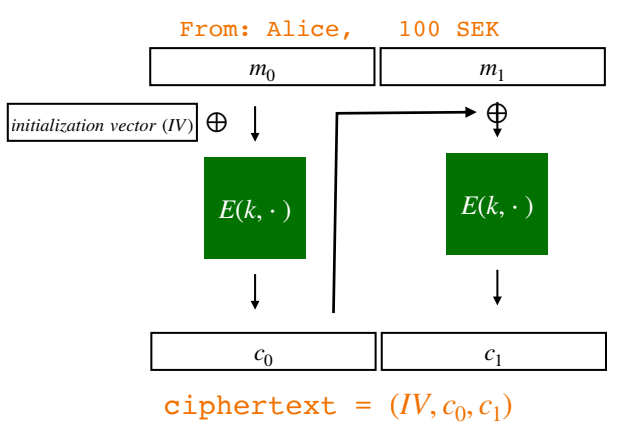
Considering the following motivating example to explain why integrity matters.

Example 1. **Fact1:** files sent over a network have well-known, predictable headers. A typical example is emails, which have a sender (From:) and receiver (To:) info, as well as date, subject and others. If the encryption is not good enough, maybe by changing some bits, the adversary $\mathcal{A}$ might change the destination. Changing "`From: Alice`" to "`From: Eve`" is trivial in CBC mode.
Fact2: Files are often encrypted in transit, so this information is not readable to the eavesdropping adversary.



This attack is trivial against AES (or any block cipher) in CBC mode, as it is shown in the following example. Remind: saying that an attack is *trivial* means that there exists an algorithm that runs in polynomial time and that succeeds in the attack with probability 1.

Example 2. Alice would like to send a Bob the $ciphertext = (IV, c_0, c_1)$ of the $message = (\texttt{From : Alice}, \texttt{100SEK})$ but the adversary $\mathcal{A}$ would change `From : Alice` to `From : Eve`.



In CBC mode, the adversary $\mathcal{A}$ can perform the attack by changing the ciphertext to be $ciphertext^* = (IV^*, c_0, c_1)$, where $IV^* = IV \oplus \texttt{From : Alice} \oplus \texttt{From : Eve}$. By decrypting the $ciphertext^*$, Bob will get the wrong $message^* = (\texttt{From : Eve}, \texttt{100SEK})$ because

$$\begin{aligned} m_0 &= D(k, c_0) \oplus IV^* = \\ &= D(k, c_0) \oplus (IV \oplus \texttt{From : Alice} \oplus \texttt{From : Eve}) = \\ &= (D(k, c_0) \oplus IV) \oplus \texttt{From : Alice} \oplus \texttt{From : Eve} = \\ &= \texttt{From : Alice} \oplus \texttt{From : Alice} \oplus \texttt{From : Eve} = \\ &= \texttt{From : Eve} \end{aligned}$$

The adversary $\mathcal{A}$ that launches this attack will succeed with 100% probability AND without knowing the secret key.
Encryption alone cannot detect the change, but Bob could by authenticate sender: if the message is encrypted using the secret key k that Bob shares with Alice, then it makes no sense that for the message to say 'from Eve'.

We wish to emphasize that the authentication problem is very different from the encryption problem. We are not worried about the secrecy of the message M. Our concern is whether the adversary $\mathcal{A}$ can profit by injecting new messages into the communication's stream, not whether $\mathcal{A}$ understands the contents of the communication. Encryption schemes are not designed against malleability, we need a new cryptographic primitive to provide data integrity, and this is called a Message Authentication Code (MAC).

2 Message Authentication Codes (MAC)

2.1 Definition

Definition 2. A **Message Authentication Code (MAC)** is a pair of efficient algorithms (MAC, Ver) with the following syntax:

- $MAC : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ is a probabilistic algorithm that takes in input a key k, a message m and outputs a tag t.
- $Ver : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{0, 1\}$ is a deterministic algorithm that takes in input a key k, a message m and a tag t, and returns 1 (accept) or 0 (reject).

where $\mathcal{K}$ is the key space, $\mathcal{M}$ is the message space, and $\mathcal{T}$ is the tags pace.
Moreover (MAC, Ver) should satisfy the **correctness** condition:

$$Pr[Ver(k, m, MAC(k, m)) = 1] = 1 \text{ for all } k \in \mathcal{K}, m \in \mathcal{M}$$

The MAC algorithm creates a tag t for a specific message m. Usually the length of the tag is between 32 and 128 bits. MACs of 32 bits, 64 bits, 96 bits, and 128 bits are common.
The correctness condition only says that if we try to verify the right key k, the right message m, and the right tag t created with a MAC with the same k and m, we should get 1 (accept) with probability 1.
In summary, when we verify something that is computed with MAC correctly, we should get 1 for the Ver algorithm.

Note: key generation is always trivial, pick a random key from the key space $\mathcal{K}$; the key is the same for both Alice and Bob (we are in the symmetric encryption contest).

Using MAC, how could we protect communication over an insecure channel? In other words, how do MACs prevent any third party (or the channel) from altering, changing, or manipulating the communication?

Figure 3: MAC prevent any third party $\mathcal{A}$ from altering the communication.
Alice and Bob share the same key k. Alice creates a MAC tag t for the message m.
Alice sends Bob the pair (m, t). The security of the MAC should guarantee that if the adversary $\mathcal{A}$ attempts to alter m (to $m^* \neq m$) it should also generate a new tag t^* and the probability that m^*, t^* is accepted by the verification algorithm should be negligible. We are not caring anymore about hiding the message m, but we want to prove the integrity of the message.

2.2 Existential Unforgeability under Chosen Message Attack (EUF-CMA)

The aim of the next security notion is to quantify the adversary's likelihood in forging a valid tag t^* for a new (different) message m^*. A tag t is valid for a message m against the key k, if $Ver(k, m, t) = 1$. In this contest, the $\mathcal{A}$'s goal is **none of the following**:

- to decrypt the communication –we do not care about secrecy, only about integrity;
- to recover the secret key –because it is a too strong requirement, and damage can be done with less; nor
- to modify the content of the communication –this is a ok but too vague goal, everybody can flip bits...

The correct way to formalize the security notion for MACs is to quantify the probability that the adversary succeeds in **producing a tag** for a **known message** that the receiver will deem authentic, and that is different from what has been sent during the communication. In crypto terms, we would like to formalize the notion of **Unforgeability under Chosen Message Attack**. For example, by flipping bits, the adversary $\mathcal{A}$ could modify the message, but we want that the adversary knows what new message is generated by the change. As usual, we give all possible (within reason) power to the adversary. In this case, this means we allow $\mathcal{A}$ to have black-box access to the verification algorithm so that it can check whether some forgery attempts are valid or not.

The word *forgery* comes from melting metal; the idea is that when you forge, you shape the metal as you like. We are using the word forgery if the adversary is able to *reshape* the message and the tag into something else.

To better understand the adversary, we can summarize its goals and powers as follows:

Goal: to generate a message-tag pair (m^*, t^*) such that, $\mathcal{A}$ knows m^* and m^* is different from any m honestly produced by the sender and observed by $\mathcal{A}$, and (m^*, t^*) is accepted by the verification algorithm. Such a pair is called **forgery**.

Powers: during the security game $\mathcal{A}$ is an algorithm that:

⋆ is efficient (probabilistic, and runs in polynomial time $< 2^{60}$);

⋆ can see everything transmitted over the communication channel;

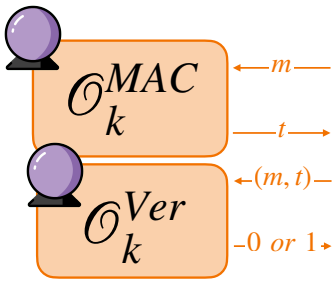
⋆ knows all details of the encryption scheme except for the secret key, in accord with Kerckhoffs's principle;

⋆ can drop, replace and inject information into the communication channel.

If the adversary has these powers it is called a **passive adversary**. If, in addition, $\mathcal{A}$ also has this power then it is called an **active adversary**, because it interacts in an *active way* with the communication channel. Moreover, to simulate prompting the sender with messages to authenticate, and testing whether a message-tag pair is a valid forgery, in the security game $\mathcal{A}$ has access to two oracles (algorithms that $\mathcal{A}$ interacts with in a black-box way):

⋆ An authentication oracle $MAC(k, \cdot)$ that on input a message m, evaluates its tag t, and returns t to the adversary.

⋆ A verification oracle $Ver(k, \cdot, \cdot)$ that on input (m, t) returns 0 or 1 according to whether the input pair is rejected or accepted as valid.



These oracles are keyed algorithms, i.e. algorithms that employ a secret key (the only thing that should stay unknown to the adversary). Since the adversary can interact with the these algorithms but cannot *see* how they operate (otherwise $\mathcal{A}$ would also observe the key), these algorithms are called *oracles*, and interactions are constrained to $\mathcal{A}$ giving an input and receiving one output.

At first glance, it may seem like an adaptive chosen-message attack is unrealistically generous to our adversary; after all, if an adversary could really obtain a valid MAC for any message it wanted, then what is the point of authenticating messages at all? In fact, there are several good arguments for allowing the adversary such a strong capability. First, we will see examples—higher-level protocols that use MACs—where adaptive chosen-message attacks are quite realistic. Second, recall our general principles. We want to design schemes that are secure in any usage, this requires that we make worst-case notions of security.

Security Game The Security for MACs is described in the following figure. The aim is to quantify the $\mathcal{A}$'s likelihood in forging a valid tag t^* for a new (different) message m^*.

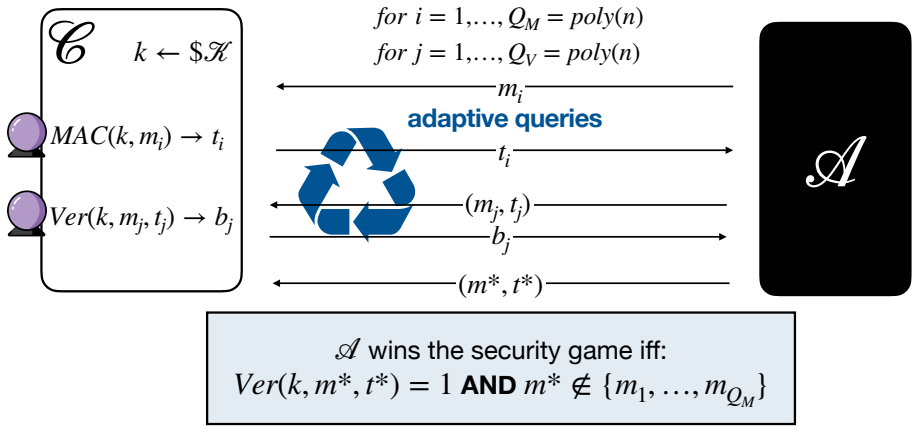


Figure 4: Unforgeability under Chosen Message Attack

The difference between this and the security games we saw thus far is that now the adversary $\mathcal{A}$ does output just a single bit. In order to attack authentication, the adversary needs to construct a valid tag for a message not seen before. This security game is called **Unforgeability under Chosen Message Attack**.

Verbose Description of EUF-CMA During the first part of adversarial attack, using the communicating parties' secret key k, the adversary $\mathcal{A}$ is able to request a MAC tag on any message that it wishes, and is able to request a verification on any pair (m_j, t_j) for the message m_j and the tag t_j that it wishes. Q_M is the number of queries for tags asked by the adversary $\mathcal{A}$, while Q_V is the number of queries for verification. Both Q_M and Q_V are polynomial in n.

At the end of this attack, the adversary $\mathcal{A}$ attempts to break the MAC scheme: the adversary $\mathcal{A}$ succeeds if it

outputs (m^*, t^*), where m^* a new message and t^* is a valid MAC tag upon that message. By a new message, we mean one that the adversary $\mathcal{A}$ did not query to the MAC tag oracle, i.e. $m^* \notin \{m_1, \ldots, m_{Q_M}\}$.
This level of security is called existential unforgeability against a chosen message attack. The "existential unforgeability" refers to the fact that the adversary $\mathcal{A}$ should not be able to generate a valid MAC tag on any message, and the "chosen message attack" refers to the fact that the adversary $\mathcal{A}$ is able to obtain MAC tags on any messages it wishes during its attack.

Definition 3. A Message Authentication Code MAC is said to be **secure (unforgeable under chosen message attack)** if for all efficient adversaries the probability that adversary $\mathcal{A}$ wins in the security game is negligible. Formally,

$$Pr[Ver(k, m^*, t^*) = 1 | (m^*, t^*) \leftarrow \mathcal{A}^{\mathcal{O}_k^{MAC}, \mathcal{O}_k^{Ver}} \wedge m^* \notin \{m_i\}_{i=1}^{Q_M}] \leq negl(n)$$

where n is the size of the key space $\mathcal{K}$.

In the previous definition, this $\mathcal{A}^{\mathcal{O}_k^{MAC}, \mathcal{O}_k^{Ver}}$ is only a compact way to mean that the adversary has access to the oracles described above.

What about the replay attack? The *replay attack* happens if I send the same message multiple times. The problem is that when we have authentication, once we have a valid tag, this tag will always be valid for the same message. MAC and signatures do not protect against replay attacks, but this problem could be easily fixed by including time stamps to make messages unique (and do not accept 'old' messages).

2.3 The Raw CBC-MAC Attack

This is an example of MAC constructed from a block cipher E run in CBC mode. We do not have IV because we are no longer caring about secrecy, we want only a short proof that the received message is not changed and equals the sent message. The random IV in CBC encryption mode serves to prevent a dictionary attack on the first ciphertext block. Confidentiality is not a concern for MACs, so $IV = 0$ is good enough. The 'Tag' is only one block long (so usually shorter than a message, that can be multiple blocks long).

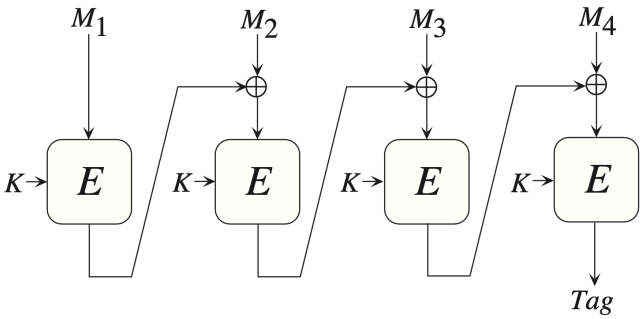


Figure 5: Example of RAW version of CBC-MAC works.

Fig. 5 depicts the raw CBC-MAC construction. p We will show that this MAC is not unforgeable. Fig. 6 shows a possible attack that exploits a message extension.

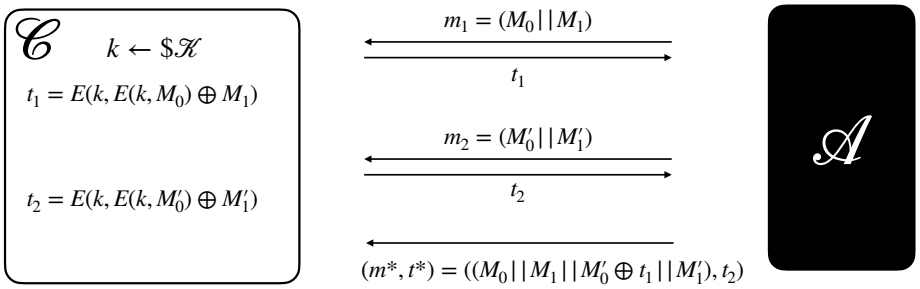


Figure 6: RAW CBC-MAC message extension attack

In the RAW CBC-MAC message extension attack:

- the challenger $\mathcal{C}$ picks at random a key k;
- the adversary $\mathcal{A}$ is querying for a message $m_1 = (M_0||M_1)$, two blocks long;
- then the adversary $\mathcal{A}$ is querying for a second message $m_2 = (M_0'||M_1')$, two blocks long;
- finally, the adversary $\mathcal{A}$ returns a forgery that is a longer message and tag (m^*, t^*) (here the name extension attack).

Note that in order to verify the pair (m^*, t^*) the verification algorithm would run: $MAC(k, m^*)$ to obtain a tag t' and the return 1 if and only if $t^* = t'$. However, m^* is crafted so that $MAC(k, m^*)$ returns exactly $t^* = t_2$, indeed

$$\begin{aligned} MAC(k, m^*) &= E(k, M_1' \oplus E(k, M_0' \oplus t_1 \oplus E(k, M_1 \oplus E(k, M_0)))) \\ &= E(k, M_1' \oplus E(k, M_0' \oplus t_1 \oplus t_1)) \\ &= E(k, M_1' \oplus E(k, M_0')) \\ &= t_2 \\ &= t^* \end{aligned}$$

We can mitigate the attack in the following way described in the picture, just adding the xoring with a key in the last round (different key for the case where padding is needed):

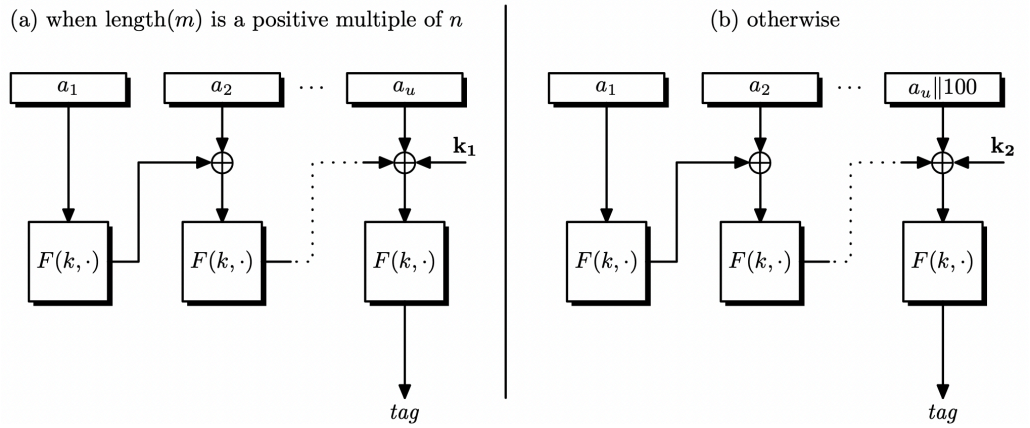


Figure 7: ANSI CBC-MAC

2.4 MAC vs Hash Functions

Cryptographic hash functions are usually faster to compute than block ciphers, in software implementations. The code that implements many hash functions is free, ready to use and can "cross borders" (USA used to restrict the export of cryptographic technologies and devices until 1992!)
The problem is that hash functions are not designed for message authentication, and usually do not have keys! How to go about this? HMAC for internet security (TLS1.2, SSH).

HMAC is a special construction of a MAC that uses a key and hash functions, so it is very fast to compute.

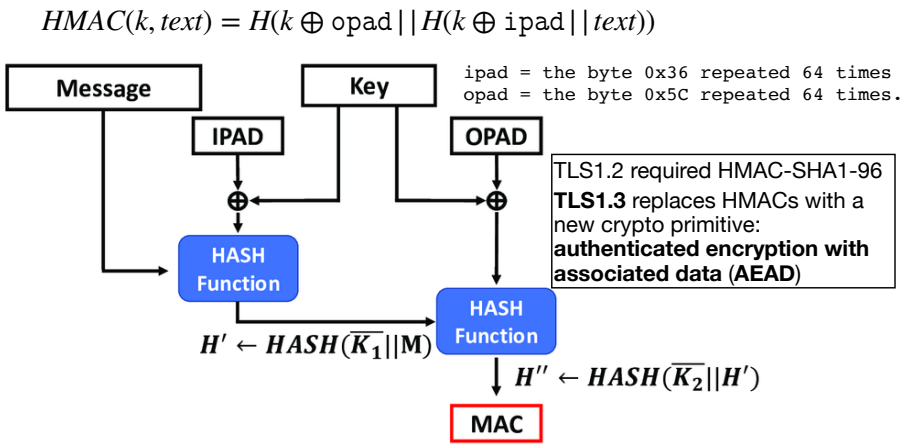


Figure 8: How HMAC works based on a hash function $HASH$

The details of how HMAC works are not of interest for this course.

3 Authenticated Encryption

This section is the culmination of our first module of the course that presents symmetric cryptography. Here we construct systems that ensure both data secrecy (confidentiality) and data integrity combining encryption and MAC.
We can obtain authenticated encryption via generic composition of a block cipher and a MAC. We present two examples.

3.1 Encrypt-And-MAC

We can encrypt our data and then MAC, in this way we should have confidentiality and integrity.

Encrypt-and-MAC:

AE.Encrypt	AE.Decrypt
Split $k = (k_0 \| \| k_1)$	Split $k = (k_0 \| \| k_1)$
$c_0 \leftarrow E(k_0, m)$	$m \leftarrow D(k_0, c_0)$
$c_1 \leftarrow MAC(k_1, m)$	$b \leftarrow Ver(k_1, m, c_1)$
return $c = (c_0, c_1)$	if $b = 1$ return m
	Else return $\perp$

The **Encryption** function takes the key k, this is a longer key build from the concatenation of two keys k_0 and k_1.
The ciphertext is built by two different parts:

- c_0 is the encryption of the message m using key k_0;
- c_1 is tag of the message m obtained using the other key k_1;

For **Decryption**, first we split the key k as before. We need first to decrypt the message, then to verify the integrity of the tag.

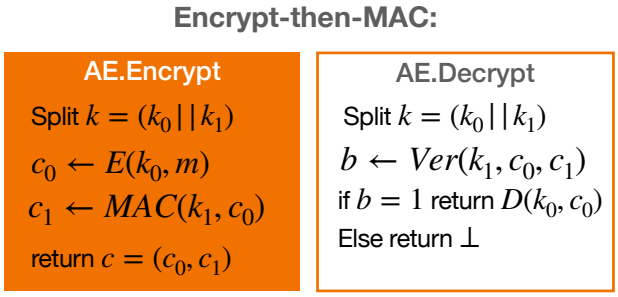
Encrypt-and-MAC is a way to achieve integrity and encryption, but this is not the best way to do it because:

- c_1 may leak information about m and
- decryption happens before the integrity check. This could be a problem because if what we are decrypting turns out not to be correct, then we have done computation for nothing and even worse, what if the unauthenticated message is a malware? We decrypt it, and then we read/run it for verification.

It would be much better if we could check integrity prior to decrypting.

3.2 Encrypt-Then-MAC

This procedure is the most secure way to compose the two primitives. It is used in TLS1.3, IPsec, GCM.



The **Encryption** function takes the key k, this is a longer key build from the concatenation of two keys k_0 and k_1.
The ciphertext is built by two different parts:

- c_0 is the encryption of the message using key k_0;
- c_1 is the tag of the **ciphertext** c_0 (not anymore the MAC of the plaintext message) using the other key k_1;

For **Decryption**, first we split the key k as before. Now we can check the integrity of the ciphertext before decrypting the message.

3.3 Galois Counter Mode (GCM)

The Galois Counter Mode (GCM) is one of the recommended block cipher modes of operation by the NIST since 2007, which has made GCM and GMAC official standards.

In order to have a quick overview, the **Galois Counter Mode (GCM)**

- is an authenticate encryption scheme of the **Encrypt-then-MAC** style;
- is widely adopted for its **great performance**;
- provides data authenticity (**integrity**) and **confidentiality** simultaneously;
- may also authenticate plaintext Associated Data (AEAD), e.g., headers.

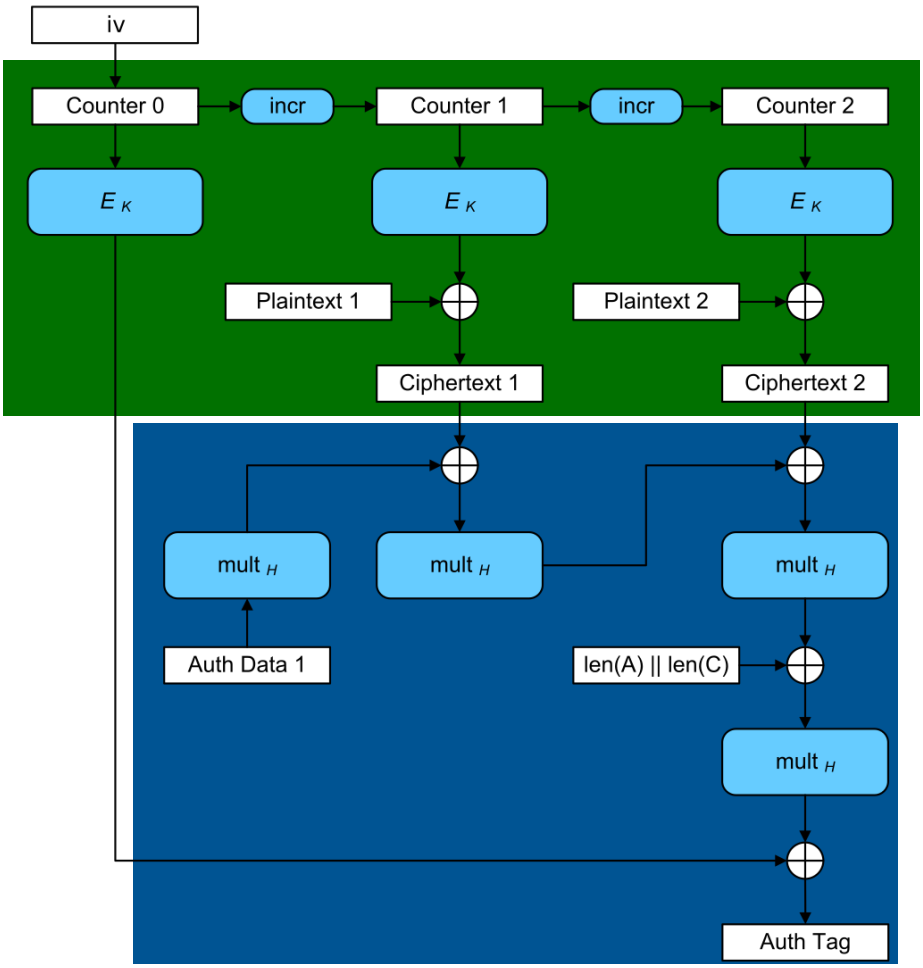


Figure 9: How GCM works.
In the green part there is the Encryption part. It is a kind of CTR: there is some IV and a counter. E_k is the AES.
In the blue part there is the MAC part.
Here, $mult_H$ is a bit multiplication with a fixed string.

Aside on Galois Field Multiplication (for crypto aficionados) The GCM uses Galois Field Multiplication (the mult_H box in the GCM figure). Our aim is to find a way to multiply bit strings together. We could multiply them bit by bit (component by component), but this is not convenient since we will quickly end up with a lot of zeros.
The intuition is

- see bit-strings as vectors with coefficients over $\mathbb{Z}_2 = \{0, 1\}$;
- see vectors as polynomials, for example for bit strings of size 128,

$$(b_0, b_1, \ldots, b_{127}) \quad \Longleftrightarrow \quad \sum_{i=0}^{127} b_i x^i \quad = \quad b_0 + b_1 x + \cdots + b_{127} x^{127}$$

- we already know how to multiply polynomials.

In order to make sure the result of the multiplication is always a bit-string of length 128 we need to do operations in a special mathematical object called **Galois Field**. This is the Galois Field used in GCM:

$$GF(2^{128}) := \mathbb{Z}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

In simple terms, in this field we substitute x^{128} with $x^7 + x^2 + x + 1$ modulo $\mathbb{Z}_2$.

Example 3. Give $f(x) = x^{100} + x + 1$ and $g(x) = x^{28} + x^2$, compute the value of the multiplication if $f(x)$ and $g(x)$ in the Galois Field used in GCM $GF(2^{128}) := \mathbb{Z}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$.

$$\begin{aligned} f(x) \cdot g(x) &= (x^{100} + x + 1) \cdot (x^{28} + x^2) = \\ &= x^{128} + x^{102} + x^{29} + x^3 + x^{28} + x^2 = \\ &= (x^{128}) + x^{102} + x^{29} + x^3 + x^{28} + x^2 = \\ &= (x^7 + x^2 + x + 1) + x^{102} + x^{29} + x^3 + x^{28} + x^2 = \\ &= x^{102} + x^{29} + x^{28} + x^7 + x^3 + x + 1 \end{aligned}$$

If you want to read up on this (optional):
https://eprint.iacr.org/2004/193.pdf,
https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Group Theory, DH, Security Assumptions

Lecturer: Elena Pagnin Lecture 5 on Nov 19

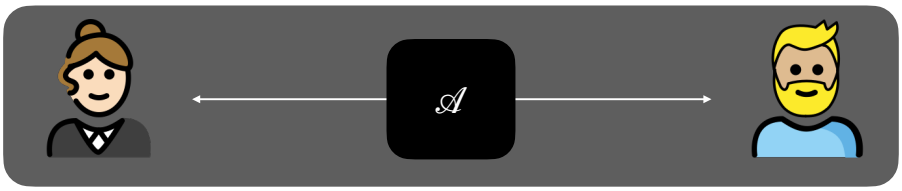
Scribes: Willem Brahmstaedt, Hanna Ek & Lucia Lavagnino Last update December 13, 2024

Contents

1 Key-Exchange

1.1 Problem Statement

The goal of asymmetric cryptography is to enable secure communication between two actors who have no previously shared secret key. How could Alice and Bob agree on a shared key? How could they share a key securely? If they have a error-free transmission channel through which they can communicate in a perfectly secure way, they could directly use such channel to communicate the message.
The problem is, how do Alice and Bob create this channel? They need to agree on the same key for securing the communication, but how can they agree on the same secret key using only a public channel, that is error-free (no accidental bit flips due to communication error) but not secure, and is monitored by the adversary? Our goal is to find a way for Alice and Bob to share a secret key k without relying on a previously shared secret using a public channel that is monitored by the adversary $\mathcal{A}$, which for now is passive, i.e., an eavesdropper. They could not rely on a previously shared secret because there is always the problem: how could they share the previous secret?



1.2 A Simple Solution

As a first approach, consider that Alice and Bob can use exponentiation, that is, they will exchange numbers obtained as $g^{\text{something}}$. This very basic protocol for key exchange is described in the following figure:

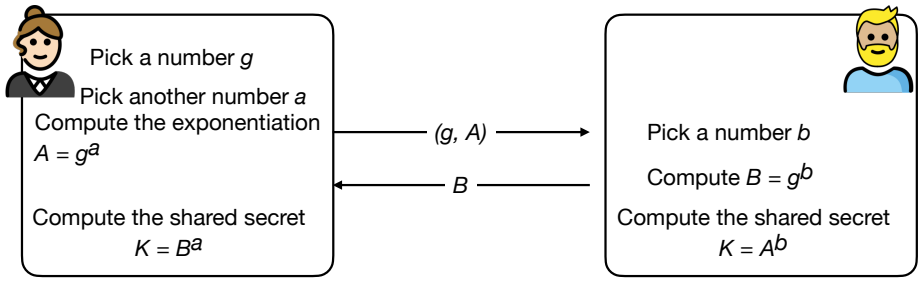


Figure 1: Diffie-Hellman Key Exchange Protocol

The idea in Figure 1: we have two parties that want to communicate, Alice and Bob.

- Alice chooses two random numbers g and a;
- Alice rises g to the power of a and obtains $g^a = A$. She then sends (g, A) to Bob;
- Bob chooses a random number b and calculates $g^b = B$ and sends B to Alice.

They both now have the means to calculate the shared secret: $K = B^a$ and $K = A^b$, respectively. K is the same for both Alice and Bob since

$$K = B^a = (g^b)^a = g^{ba} = g^{ab} = (g^a)^b = A^b = K.$$

But what prevents $\mathcal{A}$ from learning K given (g, A, B)? As it is right now, nothing unless we restrict the computational power of $\mathcal{A}$! In practice, in order to make it computationally hard to resolve K, we have to do the key exchange in the correct domain. If g is prime and a, b are large positive integers, it could work, meaning that finding x such that $X = g^x$ given only g and X is "hard" (in principle one would need to compute all the powers of the prime g to find which x yields X).

Example 1. Let $g = 7$ and $A = 30226801971775055948247051683954096612865741943$. What is the value of '$a$' such that $g^a = A$? This problem might look hard for humans, but a computer can efficiently figure out that $a = 55$.

The approach shown in the example could work, But this approach has no upper limit on how large A, B, K can become, and in cryptography, since the key is the only piece of information hidden from the adversary, it is fundamental to reason about a fix-size key length. This very basic protocol for key exchange is still interesting since it establishes a shared secret key, so that symmetric encryption can be used later on. Moreover, if we want to use K for a symmetric encryption scheme, K needs to be encoded into an n-bit string for some fixed value n. This situation is similar to the Galois counter mode, where we wanted to find a meaningful way to multiply two binary strings.
Let us introduce a domain that fixes the mentioned challenges.

1.3 Formalization: Group Theory

If we want that, at the end of the key exchange protocol, Alice and Bob can obtain the same key K used for a symmetric encryption scheme, K needs to be encoded into an $n-$bit string for some fixed value n. We need a *mathematical object* that allows us to do arbitrary exponentiations while guaranteeing the values we get stay within a certain range.
The domain in which this will work is in a group. The mathematical object of a group $\mathbb{G}$ is defined as following.

Definition 1. A **group** is a pair $(\mathbb{G}, \star)$ where $\mathbb{G}$ is a set of elements and $\star : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ is an operation that satisfies the following properties:

1. **Closure:** $\forall g, h \in \mathbb{G}$ it holds that $g \star h \in \mathbb{G}$, i.e. when applying $\star$ on any two elements in the group, then the result of the operation is also in the group.
2. **Associativity:** $\forall\, g, h, k \in \mathbb{G}$ it holds that $(g \star h) \star k = g \star (h \star k)$, i.e. the order of the operation does not effect the result.
3. **Identity:** $\exists\, e \in \mathbb{G}$ such that $e \star g = g \star e = g$ for all $g \in \mathbb{G}$, i.e. there is an element e in $\mathbb{G}$ which when used with another element g in $\mathbb{G}$ and $\star$ always returns g.
4. **Inverse:** $\forall\, g \in \mathbb{G}$ there exists a unique element $\bar{g} \in \mathbb{G}$ such that $g \star \bar{g} = \bar{g} \star g = e$, i.e. an element combined with its inverse in $\star$ is the identity element.

The operation $\star$ for cryptography purpose is usually $+$ (the *classical* addition) or $\cdot$ (the *classical* multiplication). An example of a group is $\mathbb{G} = (\mathbb{Z}_p, +)$, where $\mathbb{Z}_p = \{0, 1..., p-1\}$ is the set, and $g \star h = g + h \mod p$ is the operation. Another example of a group that one encounters daily is the analog clock, with the set consisting of the hours and addition mod 12 the operation.

Example 2. In $\mathbb{Z}_7$, we can compute $3 + 5$ and 3^5 as following:
$3 + 5 = 1 \mod 7$
$3^2 = 9 = 2 \mod 7 \qquad 3^4 = 3^2 \star 3^2 = 2 \star 2 = 4 \mod 7 \qquad 3^5 = 3 \star 3^4 = 3 \star 4 = 12 = 5 \mod 7$

In many cases the symbol $\mathbb{G}$ is used to referred interchangeably both to the set of elements and the actual group (with an operation and satisfying all four properties).

There are some important concepts about groups:

- An element $g \in \mathbb{G}$ is called a **generator** for $\mathbb{G}$ if all elements in the group can be seen as "multiple" of g, i.e., $\langle g \rangle = \{g, g \star g, g \star g \star g, \dots\} = \mathbb{G}$.
- A group $\mathbb{G}$ that admits a **generator** $g \in \mathbb{G}$ is called a **cyclic group**.
- A group $(\mathbb{G}, \star)$ is **commutative** (abelian) if for any two elements $g, h \in \mathbb{G}$ it holds that $g \star h = h \star g$.
- The **order** of a group is equal to the number of elements in the group (its cardinality), i.e., $ord(\mathbb{G}) = |\mathbb{G}|$. The order of an element $h \in \mathbb{G}$ is the smallest positive integer $d = ord(h) > 0$ such that $\underbrace{h \star h ... \star h}_{d\ times} = e$, where $e \in \mathbb{G}$ is the identity of the group.

Note: not all groups are cyclic, there exist groups that do not admit a generator. However, in this course we will only see groups that are commutative, and almost only work with cyclic groups.

1.3.1 Important results in group theory

There are several important theoretical results about groups, here we collect the ones that are most relevant for this course, and give the (informal) statements without proofs.

- All cyclic groups are commutative.
- The generator is not unique (if a group admits a generator, then it admits multiple generators).
- If a group has order $n > 1$ then for all $g \in \mathbb{G} : \underbrace{g \star g \cdots \star g}_{n\ times} = e$ (where e denotes the neutral element).
- For every $h \in \mathbb{G}$ it holds that $ord(h)$ divides $ord(\mathbb{G})$.

1.3.2 More (useful) examples of Groups

Example 3. Let $N > 1$ be an integer, then $(\mathbb{Z}_N, +)$ is a group, indeed, $\forall n, m, k \in \mathbb{Z}_N$:

- **closure:** $n + m \mod N$ is still an element of the group $\mathbb{Z}_N$;
- **associativity:** $(n + m) + k = n + (m + k) \mod N$;
- **identity:** $0 + n = n + 0 = n \mod N$;
- **inverse:** the inverse of n is $N - n$

Example 4. Let $N > 1$ be an integer, then $(\mathbb{Z}_N, \cdot)$ is **not** a group. It suffices to show that $(\mathbb{Z}_N, \cdot)$ does not satisfy the definition of group. For instance: the element 0 is in $\mathbb{Z}_N = \{0, 1, 2, \ldots, N-1\}$ but has no multiplicative inverse, indeed, $\forall n \in \mathbb{Z}_N$, $n \cdot 0 = 0 \mod N$.

Example 5. In $(\mathbb{Z}_7, +)$, the order of the element 1 is 7 because $1 + 1 + 1 + 1 + 1 + 1 + 1 = 7 = 0 \mod 7$. In $(\mathbb{Z}_7^*, \cdot)$, the order of the element 1 is 1 because $1 = 1 \mod 7$.

Next, let us condsider $\mathbb{Z}_N^* = \{x \in \{1, 2, ..., N-1\} |\ \gcd(x, N) = 1\}$, this is the set of all elements in $\mathbb{Z}_N$ that have a multiplicative inverse (modulo N). To prove this, one can use the extended Euclidian algorithm[1] to compute $s, t \in \mathbb{Z}$ such that $x \cdot s + N \cdot t = \gcd(x, N) = 1$, and by taking the equality modulo N we get $x \cdot s = 1 \mod N$, i.e. $x^{-1} = s \mod N$. Hence the extended Euclidian algorithm gives a way to compute an inverse of x for any x in $\mathbb{Z}_N^*$. This means that $\mathbb{Z}_N^*$ is indeed the set of all invertible elements (numbers) in $\{0, 1, 2, \ldots, N-1\} = \mathbb{Z}_N$. This proves that the two stated sets are equivalent:

$$\begin{aligned} \gcd(x, N) &= 1 \\ &\Downarrow \quad \text{By the extended Euclidian algorithm} \\ x \cdot s + N \cdot t &= 1 \\ &\Downarrow \quad \mod N \\ x \cdot s &= 1 \end{aligned}$$

the number s is the multiplicative inverse of x in $\mathbb{Z}_N^*$.

Example 6. Let $N > 1$ be an integer, then $(\mathbb{Z}_N^*, \cdot)$ is a group.

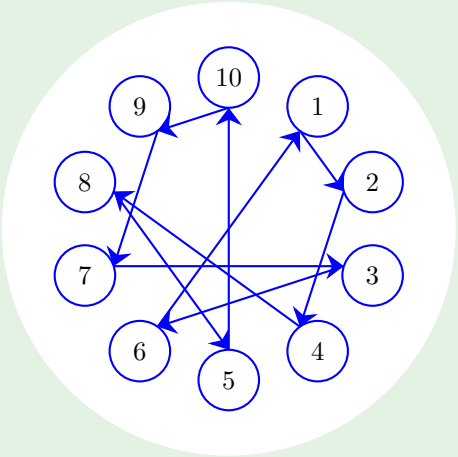
For Cryptography purpose, the cyclic group with a generator are the most important.

[1] See here for more details: https://en.wikipedia.org/wiki/Euclidean_algorithm

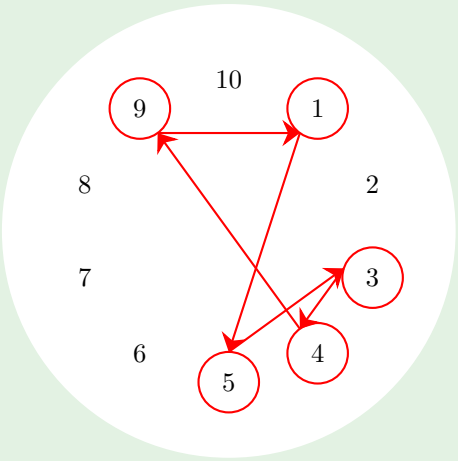
Example 7. Consider the group $(\mathbb{Z}_{11}^*, \cdot)$ where $\mathbb{Z}_{11}^* = \{1,2,3,4,5,6,7,8,9,10\}$ since every element (apart from 0) is coprime with 11. Let us study the behaviour of some of the elements in the group. The notation $\langle h \rangle$ - menas the group generated by the element h.

$$\langle 1 \rangle = \{1\}$$

$$\begin{aligned}
\langle 2 \rangle = \Big\{ & 2^0 = 1 \mod 11, \\
& 2^1 = 2 \mod 11, \\
& 2^2 = 4 \mod 11, \\
& 2^3 = 8 \mod 11, \\
& 2^4 = 16 = 5 \mod 11, \\
& 2^5 = 2 \cdot 2^4 = 2 \cdot 5 = -1 = 10 \mod 11, \\
& 2^6 = 2 \cdot 2^5 = 2 \cdot -1 = -2 = 9 \mod 11, \\
& 2^7 = 2^2 \cdot 2^5 = 4 \cdot -1 = -4 = 7 \mod 11, \\
& 2^8 = 2^3 \cdot 2^5 = 8 \cdot -1 = -8 = 3 \mod 11, \\
& 2^9 = 2^4 \cdot 2^5 = 5 \cdot -1 = -5 = 6 \mod 11, \\
& 2^{10} = 2 \cdot 2^9 = 2 \cdot 6 = 12 = 1 \mod 11 \Big\} \\
= & \{1,2,4,8,5,10,9,7,3,6\}
\end{aligned}$$



$$\begin{aligned}
\langle 5 \rangle = \Big\{ & 5^0 = 1 \mod 11, \\
& 5^1 = 5 \mod 11, \\
& 5^2 = 25 = 3 \mod 11, \\
& 5^3 = 5 \cdot 5^2 = 5 \cdot 3 = 15 = 4 \mod 11, \\
& 5^4 = 5^2 \cdot 5^2 = 3 \cdot 3 = 9 \mod 11, \\
& 5^5 = 5 \cdot 5^4 = 5 \cdot 9 = 45 = 1 \mod 11 \Big\} \\
= & \{1,5,3,4,9\}
\end{aligned}$$



Note that:
The element 1 is NOT a generator of $\mathbb{Z}_{11}^*$.
The element 2 generates the whole group $\mathbb{Z}_{11}^*$ i.e. $|\langle 2 \rangle| = 10 = |\mathbb{Z}_{11}{}^*|$, the element 2 is therefore a generator in $\mathbb{Z}_{11}^*$.
The element 5 is NOT a generator, as $|\langle 5 \rangle| = 5$, only generates a subgroup of order 5.

1.3.3 Euler's Totient function

A useful function in cryptography is **Euler's Totient** function (also known as Euler's φ), $\varphi : \mathbb{N} \to \mathbb{N}$, which given as input a positive integer N returns the number of positive integers smaller than N that are relatively prime to N. In other words, the Euler's Totient function of N give us the number of elements in $\mathbb{Z}_N^*$, since this group contains all the numbers coprime with N: $|\mathbb{Z}_N^*| = \varphi(N)$. This is computed as:

$$\varphi(N) = N \prod_{p|N} \left(1 - \frac{1}{p}\right).$$

where $p|N$ means that the number p divides the number N, i.e. the Euclidean division between N and p has a remainder equal to 0.
Useful re-formulations:

$$\begin{aligned}
\varphi(p) &= p - 1 && \text{where } p \text{ is a prime} \\
\varphi(p^n) &= (p-1) \cdot p^{n-1} && \text{where } p \text{ is a prime} \\
\varphi(N) &= (p-1)(q-1), && N = pq \text{ where } p, q \text{ are primes.} \\
\varphi(p_1 \cdot p_2 \cdots p_k) &= \varphi(p_1) \cdot \varphi(p_2) \cdots \varphi(p_k), && \text{where } p_1, p_2, \ldots, p_k \text{ are primes.}
\end{aligned}$$

Next, we list some important results concerning Euler's φ function. The statements are a bit informal (and we will not be proven) but they are formulated in a way that should aid solving the exercises.

Theorem 1 (**Euler's Theorem**). For all $a \in \mathbb{Z}_N^*$ it holds that $a^{\varphi(N)} = 1 \mod N$.

Theorem 2 (Corollary 1 **Fermat's Little Theorem**). Let $p > 2$ be a prime number, then $\forall a \in \mathbb{Z}_p^*, a^{p-1} = 1 \mod p$.

Theorem 3 (Corollary 2). Let $N > 2$ be a positive integer. For all $a \in \mathbb{Z}_N^*$ and $x \in \mathbb{N}$, it holds that $a^x = a^{x'} \mod N$, where $x' = x \mod \varphi(N)$.

The last corollary is important because in general, if we want to raise a number to a large exponent, it is convenient to firstly compute the exponent modulo the Euler's function, and then proceed with computing the exponentiation. The last corollary is a consequence of the Euler's theorem since, if $x' = x \mod \varphi(N)$, it means that $x' = x + k \cdot \varphi(N)$, so

$$a^{x'} = a^{x+k\cdot\varphi(N)} = a^x \cdot ak \cdot \varphi(N) = a^x \cdot \left(a^{\varphi(N)}\right)^k \underbrace{=}_{\text{Euler's Theorem}} a^x \cdot 1 = a^x \mod N$$

Example 8. How can we use these results to compute $7^{32} \mod 11$ without a calculator?
First we can notice that $\varphi(11) = 10$, hence

$$7^{32 \mod \varphi(11)} \underbrace{=}_{\text{Corollary 2}} 7^{32 \mod 10} = 7^2 = 49 = 5 \mod 11$$

2 Diffie-Hellman Key Exchange (DH)

The Diffie-Hellman key exchange protocol (DH) is one of the earliest practical examples of public key exchange implemented within the field of cryptography. Published in 1976 by Diffie and Hellman, this is the earliest publicly known work that proposed the idea of a private key and a corresponding public key. A variation of DH is implemented also in the signal protocol used in WhatsApp.

2.1 The Textbook DH Protocol

The Diffie-Hellman key exchange protocol (DH) [Textbook] is described in the following figure:

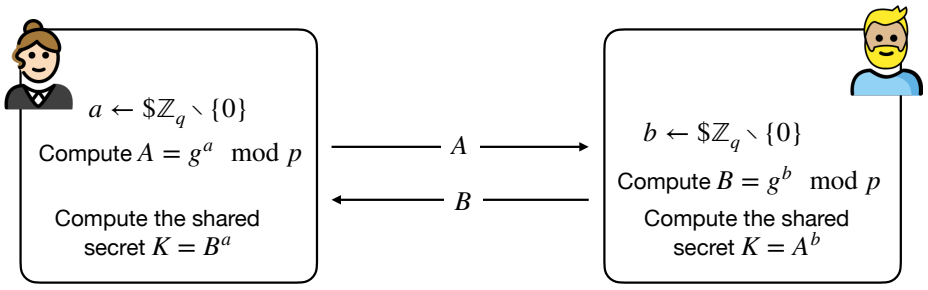


Figure 2: diffie-Hellman key exchange protocol (DH) [Textbook]

The setting is:

- let p be a large prime (2048-bits long);
- find a generator g of a subgroup of prime order $q < p$ in $\mathbb{Z}_p^*$, i.e. $|\langle g \rangle| = q$;
- let p, q, g all be public information.

Note: in $\mathbb{Z}_p^*$ there are elements that generate the whole group, but there are also elements that generated a subgroup (see Example 7).

Then, the protocol works in this way:

- Alice chooses a random number, $a \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$ and computes $A = g^a \mod p$;
- Bob chooses a random number, $b \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$ and computes $B = g^b \mod p$;
- Alice sends A to Bob, and Bob sends B to Alice;
- Alice and Bob compute the shared key as $K = B^a \mod p$ and $K = A^b \mod p$, respectively.

Now the Diffie-Hellman key exchange is complete, the shared key is the same since $K = B^a = g^{ab} = A^b \mod p$ and it will always be representable as a bit string of fix length $\log_2(p)$ (the same bitlenght as p). The values A, B, K are in the set $\mathbb{Z}_p^*$ and belong to the subgroup of $\mathbb{Z}_p^*$ of order q generated by $\langle g \rangle$. Intuitively, the security relies on the facts that, neither the values a, b not the value K should be computable from the public exchange transcript (A, B). Moreover, given A, B and g, the value K should be indistinguishable from a random element in $\langle g \rangle$. In summary, we have three reasonable way to define security:

→ from A, B, we should not be able to efficiently compute the discrete logarithms, i.e. the values a and b;

→ from A, B, we should not be able to compute the key K;

→ given A, B, the value K should be indistinguishable from random.

2.2 On the bit security of DH keys

We have just described the Textbook version of DH protocol, but that is not the more secure, it is better to apply an hash function.

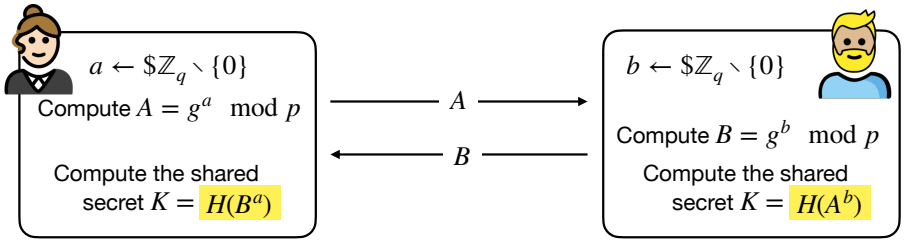


Figure 3: Diffie-Hellman Key Exchange Protocol

This is best to hash/process the textbook DH key prior to use, for a number of reasons:

1. it is (computationally) easy to find the least significant bit (LSB) of K;
2. the discrete logarithm $dLog_g(K)$ is even iff K is quadratic residue in $\mathbb{Z}_p^*$;
3. it is easy to compute the s LSBs or MSBs of K when $p - 1 = 2^s \cdot q$, with q odd. [2];
4. computing the $s + 1$-th LSB or MSB of K, when $p - 1 = 2^s \cdot q$, with q odd is hard;
5. heuristically $H(A^b)$ is a good key if $H : \{0,1\}^{|p|} \to \{0,1\}^{256}$ is a cryptographic hash function.

There are several ways of boosting security for DH and dLog, such as Triple Diffie-Hellman (X3DH), or using a group $\mathbb{G}$ which consists of points on an elliptic curve.

2.3 MiM Attack

Let us now see an attack against the textbook DH protocol. Creating a Man-in-the-Middle attack against the DH key exchange is somewhat trivial, as the adversary $\mathcal{A}$ creates one shared key with Alice and a different one with Bob. $\mathcal{A}$ then gets total control over the communication between Alice and Bob. This attack is enabled by DH's inability to authenticate whom you are doing a key exchange with. How to enable authentication will be covered later, using digital signatures.

Alice and Bob want to find a way to share a secret key k without relying on a previously shared secret AND they want to do so, using a public channel that is monitored by the Adversary (for now, we only consider passive $\mathcal{A}$ (eavesdropper)).

The **Man In the Middle Attack (MiM)** against the DH Key Exchange is described in the following figure:

[2] see ("Hardness of Distinguishing the MSB or LSB of Secret Keys in Diffie-Hellman Schemes" [FPSZ06]) https://www.di.ens.fr/ stern/data/St115.pdf for more details

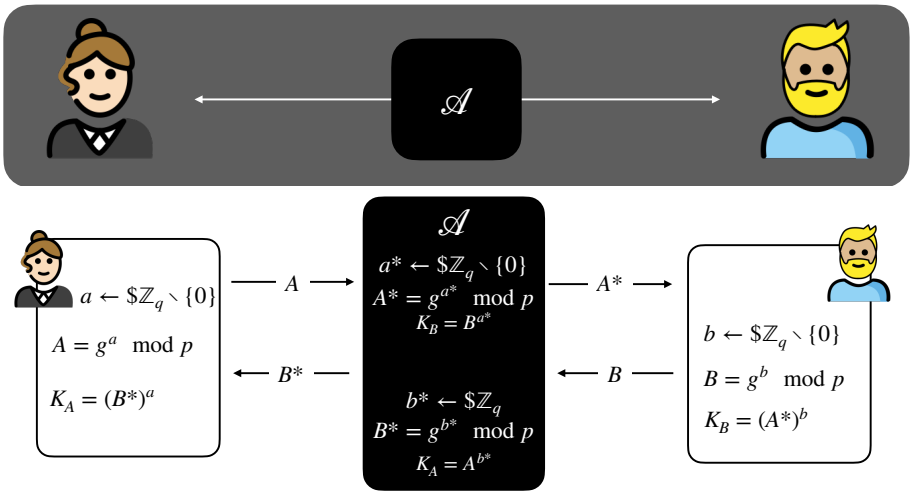


Figure 4: MiM against the DH Key Exchange

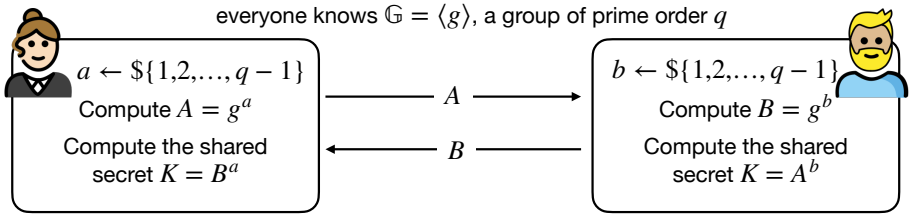
In simple terms:

- first, Alice and Bob do what they are supposed to do in the protocol: sample random exponents $a, b \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$ respectively, compute $A = g^a \mod p$ and $B = g^b \mod p$ respectively, and send A and B to the adversary $\mathcal{A}$;
- the adversary $\mathcal{A}$ can pretend to be Bob with Alice: samples a random exponent $b^* \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$, computes $B^* = g^{b^*} \mod p$, and send B^* to Alice.
  In this way, Alice and the adversary $\mathcal{A}$ have negotiate a key $K_A = (B^*)^a = (A)^{b^*}$ that only they know. We can call this key K_A since this key will never reach Bob;
- the adversary $\mathcal{A}$ also can pretend to be Alice with Bob: sample a random exponent $a^* \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$, computes $A^* = g^{a^*} \mod p$, and send A^* to Bob.
  In this way, also Bob and the adversary $\mathcal{A}$ have negotiate a key $K_B = (A^*)^b = (B)^{a^*}$ that only they know.

Whenever Alice is encrypting a message, she will use the key K_A and the adversary $\mathcal{A}$ will be able to describe the message. Then the adversary $\mathcal{A}$ can modify the message and use the key K_B to encrypt the modified message and sent it to Bob. Basically, the adversary $\mathcal{A}$ is sitting in the middle (here the name of the attack), he is reading the conversation and potentially changing them.
What is enabling this attack? DH does not authenticate whom you are doing a key exchange with. A simple solution could be to add a signature to the message to ensure the correctness of the sender.

3 Security for DH

3.1 DL, CDH, DDH Problems



Consider the definition of DH presented in the previous section. In this section we assume that q is a $n-$bit long prime.
The first attempt at a security notion for DH can then be:

Attempt 1: It is necessary that the values a, b are not obtainable from A, B.

This is a good way to define security and is really strong, is is called: the **Discrete Logarithm (DL) Problem**:

Definition 2. Let $\mathbb{G}$ be a group of prime order q, for large prime q ($\log_2(q) = n$) and g be a generator of $\mathbb{G}$.
Given $A \in \mathbb{G}$, the **Discrete Logarithm Problem (DL)** asks to find a positive integer a such that $A = g^a$ in $\mathbb{G}$, i.e find $a = log_g(A)$.

The formal definition of the Discrete Logarithm Assumption is:

Definition 3. Let $\mathbb{G}$ be a cyclic group of order q (where q is a n-bit long prime) and g be a generator of $\mathbb{G}$. The **Discrete Logarithm Assumption** states that it is computationally infeasible for any efficient attacker to find the exponent x such that $g^x = h$ for a random element $h \in \mathbb{G}$. Formally,

$$Pr\left[x^* = x | x \xleftarrow{\$} \mathbb{Z}_q, x^* \leftarrow \mathcal{A}(q, g, g^x)\right] < negl(n)$$

The last formula states that if the adversary $\mathcal{A}$ sees the prime q, the generator g and g^h for a random exponent, the probability that the adversary $\mathcal{A}$ outputs a guess for the discrete logarithm is negligible. In simple terms, the discrete logarithm assumption states that DL problem is hard.
Note that this is an **assumption** it **cannot be proven!** Decades of cryptanalysis and scrutiny by the cryptographic community world-wide has given confidence that this assumption is true, for *large enough* primes. Such an assumption is therefore considered to have **computational security.**

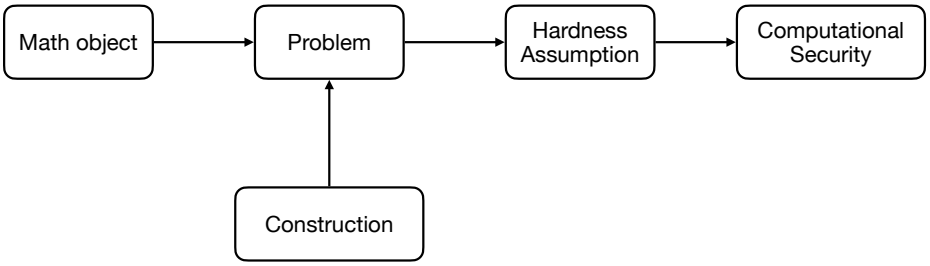


Figure 5: The flow chart of how cryptographic schemes are born. Usually everything starts from a mathematical object (e.g., a group $\mathbb{G}$), then find a problem (e.g., the DL problem). Once that you have a problem, you can build a hardness assumption. From the hardness of a problem, you could have computational security for a cryptographic construction (e.g., DH key exchange).

However, are we sure that no (efficient) adversary can compute K combining A and B but without learning a, b, i.e., without breaking DL? The following attempt to defining security for DH rules out this possibility:

Attempt 2: It should be infeasible to compute K combining A, B.

This attempt introduces the **computational Diffie-Hellman Problem (CDH):**

Definition 4. Let $\mathbb{G}$ be a group of prime order q for large prime q (i.e. $\log_2(q) = n$) and g be a generator of $\mathbb{G}$. Given $A = g^a, B = g^b \in \mathbb{G}$, the **computational Diffie-Hellman Problem (CDH)** asks to find $K \in \mathbb{G}$ such that $K = g^{ab}$.

With *large prime*, we mean that $\log_2(q) = n$.
A desicional version of the second attempt is:

Attempt 3: It should be infeasible to distinguish whether K or $K' \neq K$ was used as a shared key.

This final attempt introduces the **Decisional Diffie-Hellman Problem (DDH):**

Definition 5. Let $\mathbb{G}$ be a group of prime order q for large prime q ($\log_2(q) = n$) and g be a generator of $\mathbb{G}$. Given $A = g^a, B = g^b, C \in \mathbb{G}$, the **Decisional Diffie-Hellman Problem (DDH)** asks to determine whether $C = g^{ab}$ or not.

In summary, there are three possible ways to argue about the security of the DH key exchange protocol, each way introduces a different problem and has an accompanying security assumptions (which states that solving the problem is computationally infeasible for any efficient adversary, commonly summarized in "hard").

3.2 Relation between problems

Definition 6. Let P_1 and P_2 be two computational problems. P_1 is said to **efficiently reduce** to P_1, written $P_1 \leq P_2$ if:

1. there exists an algorithm that solves P_1 using an algorithm that solves P_2, and
2. this algorithm runs in polynomial time if the algorithm for solving P_2 does.

Proof structure: build a **reduction** (sequence of steps, program):

- assume we have an oracle (or efficient algorithm) to solve problem P_2;
- we then use this oracle to design an efficient algorithm for solving problem P_1.

We have a reduction when a problem *reduces* to another problem. When we would like to prove that there are relations between assumptions, first, we assume to have a black box oracle that knows how to solve the problem P_2, then we show a way to solve problem P_1 (by using this black box). This means that the problem P_1 reduces to P_2.
The relations between the three problems can be written as: DDH $\leq$ CDH $\leq$ DL (for more details, see[3]). These relations can be shown by assuming that there exists an efficient algorithm for solving DL, i.e., finding a such that $A = g^a$. Then, to solve CDH one can use this result and calculate the key $B^a = g^{ab} = K$. DDH can be solved in the same manner by first calculating K from A, B and then checking if $K = C$.

Next we will prove that CDH reduces to the DL problem, in other words, the hardness of the CDH problem implies the hardness of the DL problem.

Theorem 4 (CDH $\leq$ DL)**.** The *DL* problem (relative to a group $\mathbb{G}$) is at least as hard as the *CDH* problem in the same group.

Proof. Let $\mathbb{G}$ be a cyclic group of prime order q, with g a generator of $\mathbb{G}$ and q a $n-$bit long prime number. We recall the statements of the two problems:

DL (discrete logarithm): Given $g, X \in \mathbb{G}$, find $x \in \mathbb{Z}_q$ such that $X = g^x$ in $\mathbb{G}$.

CDH (computational Diffie-Hellman) Given $g, X, Y \in \mathbb{G}$, find $K \in \mathbb{G}$ such that $K = g^{xy}$, where $x = dlog_g(X)$ and $y = dlog_g(Y)$.

The intuition of the proof is given in Fig. 6

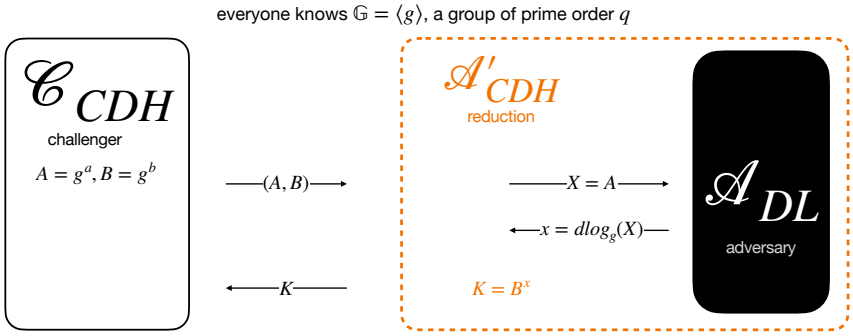


Figure 6: $CDH \leq DL$

The proof proceeds by reduction. To prove that hardness of the CDH implies hardness of the DL, we show that given black-box access to any algorithm $\mathcal{A}_{\text{DL}}$ that solves the DL problem, we can define a concrete algorithm $\mathcal{A}'_{\text{CDH}}$ that solves any instance of CDH. Let $\mathcal{A}_{\text{DL}}$ be an arbitrary PPT algorithm for the discrete-logarithm problem with respect to $(q, g, \mathbb{G})$. This means that on input $(g, \mathbb{G}, X)$, the algorithm $\mathcal{A}_{\text{DL}}$ outputs $x' \in \mathbb{Z}_q$ such that $g^{x'} = g^x$ with non-negligible probability. Note that $g^{x'} = g^x$ implies $x' = x$, since the discrete logarithm is unique (modulo q) - formally, for any generator g of the group $\mathbb{G}$ the map $(\mathbb{Z}_q, +) \to (\mathbb{G}, \cdot)$ that maps $x \mapsto g^x$ is an isomorphism. So, by assumption $\mathcal{A}_{\text{DL}}(g, \mathbb{G}, X) \to x = dlog_g(X)$ with non-negligible probability $\delta > 0$. Next, we show how to construct $\mathcal{A}'_{\text{CDH}}$. Given a CDH instance

[3] this blog bristolcrypto.blogspot.com/2014/12/52-things-number-11-what-are-dlp-cdh.html for more details

$(g, \mathbb{G}, X, Y)$ and black-box access to $\mathcal{A}_{\text{DL}}$, the reduction $\mathcal{A}'_{\text{CDH}}$ queries $\mathcal{A}_{\text{DL}}$ on $(g, \mathbb{G}, X)$ and receives back $x' \in \mathbb{Z}_q$. The reduction $\mathcal{A}'_{\text{CDH}}$ uses x' to compute $K' = Y^{x'}$ and returns K to its CDH challenger. Clearly, $\mathcal{A}'_{\text{CDH}}$ succeeds if and only if $\mathcal{A}_{\text{DL}}$ succeeds. Indeed, $K' = (Y)^{x'} = g^{yx}$ if and only if $x' = x$ (i.e., if and only if $x' = dlog_g(X)$). By assumption, the latter happens with non-negligible probability δ. Hence, $\mathcal{A}'_{\text{CDH}}$ and $\mathcal{A}$ succeed with the same probability, which means that if $\mathcal{A}_{\text{DL}}$ solves the DL problem with non-negliglbe probability δ, then $\mathcal{A}'_{\text{CDH}}$ solves the CDH problem with probability δ as well.

□

4 Public Key Cryptography (PKC)

4.1 Introduction

The introduction of public-key encryption marked a revolution in the field of cryptography. Public-key techniques enable two parties to communicate privately without having agreed on any secret information in advance. One of the fundamental concepts in Public Key Cryptography (PKC), is that a problem is *hard*, unless you know some additional information which makes solving the problem easy. This kind of *additional information*, in the Cryptography world, is called **trapdoor**. In cryptography, a problem that is hard is solvable, but requires time proportional to the age of our universe. PKC is all about this 'efficiency gap' in solving a mathematical problem.

Let us recall the definition of a one-way function (OWF):

Definition 7. A function $f : \{0,1\}^n \to \{0,1\}^d$ is **one-way** if:

- $f(\cdot)$ is efficiently computable, i.e. there exists an algorithm that computes $f(x)$ in polynomial time for all inputs $x \in \{0,1\}^n$
- for every PPT algorithm $\mathcal{A}$ there is a negligible function $negl_{\mathcal{A}}(\cdot)$ such that for sufficiently large values of $n \in \mathbb{N}$ it holds that

$$Pr[f(x) = f(x')|x \xleftarrow{\$} \{0,1\}^n, x' \leftarrow \mathcal{A}(f(x))] \leq negl_{\mathcal{A}}(n)$$

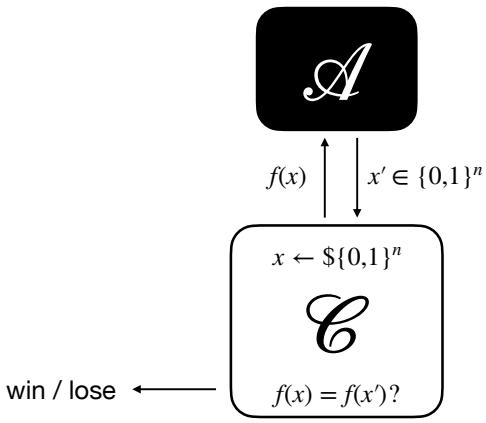


Figure 7: OWF: the challenger $\mathcal{C}$ samples a random input $x \xleftarrow{\$} \{0,1\}^n$ and sends to the adversary $y = f(x)$, the evaluation of the OWF on x. The adversary $\mathcal{A}$ is expected to send back a string $x' \in \{0,1\}^n$. $\mathcal{A}$ wins the game if its output x' is indeed a pre-image of y (hence braking the one-wayiness of f). In other words, $\mathcal{A}$ wins if it returns x' such that $f(x) = f(x')$ –remember this does not always imply that $x = x'$.

4.2 Examples of algebraic one-way trapdoor functions

Example 9. Integer Factorisation
Consider $f(p,q) = N = p \cdot q$.
Given a large composite number N it is computationally hard to find p, q (if p, q are large safe primes).

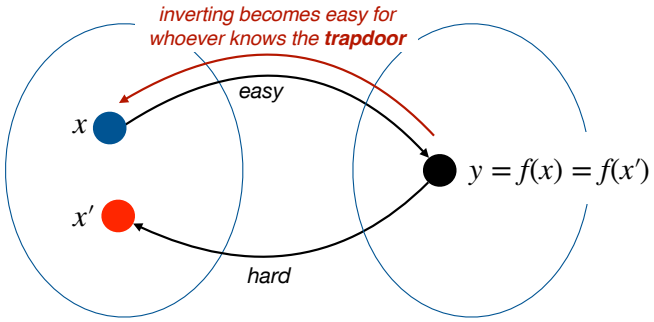
The trapdoor is (p, q).

Example 10. The RSA Assumption
Consider $f_{N,e}(x) = y \ (= x^e \mod N)$.
Given (N, e, y) such that $N = pq, log_2(p) = log_2(q) = n$, $e \in \mathbb{Z}^*_{\varphi(N)}$ and $y \xleftarrow{\$} \mathbb{Z}^*_N$ any PPT adversary has only negligible probability to find $x \in \mathbb{Z}^*_N$ such that $x^e = y \mod N$.

The trapdoor is d, the inverse of e modulo $\varphi(N) = (p-1)(q-1)$.

Example 11. Discrete Logarithm:
Consider: $f(x) = h = g^x \mod p$.
Given g a generator of $\mathbb{G}$, a prime order multiplicative group in $\mathbb{Z}^*_p$ (for a large prime p, $\log_2(p) = 2048$) and given $h \xleftarrow{\$} \mathbb{G}$, it is computationally hard to find $x = dlog_g(h)$.

The trapdoor is x.



TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Trapdoor Permutations, Digital Signatures

Lecturer: Elena Pagnin Lecture 6 on Nov 22

Scribes: Willem Brahmstaedt, Hanna Ek & Lucia Lavagnino Last update December 17, 2024

Contents

1 Public Key Cryptography (PKC)

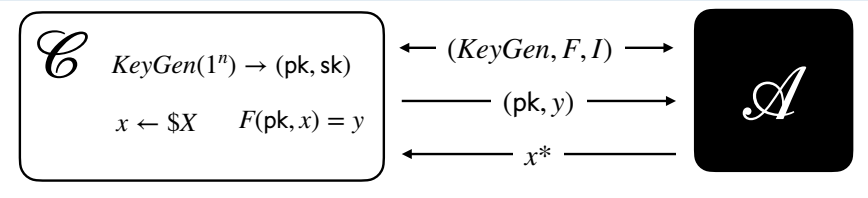
1.1 Trapdoor Permutations

A *permutation* is a specific class of functions: namely one-to-one functions from a set to another, meaning that they are bijective maps. In the following definition, F stands for *Forward algorithm* and I stands for *Inverse algorithm*. Moreover, everybody (using the pk) can compute the forward algorithm, while the inverse algorithm can only be computable by the one who knows the trapdoor (sk).

Definition 1. A **(one-way) trapdoor permutation** defined over two finite sets X, Y is a triple of PPT algorithms $(KeyGen, F, I)$ defined as follows:

- $KeyGen(1^n) \to (\mathsf{pk}, \mathsf{sk})$ is a randomized key generation algorithm;
- for every pk output by $KeyGen$, $F(\mathsf{pk}, \cdot) : X \to Y$ is efficiently computable $F(\mathsf{pk}, x) = y$;
- for every sk output by $KeyGen$, $I(\mathsf{sk}, \cdot) : Y \to X$ is efficiently computable $I(\mathsf{sk}, y') = x'$;
- $F(\mathsf{pk}, \cdot)$ is hard to invert (without the knowledge of the corresponding trapdoor, sk).

Formally, we have the following security game:



Describing the above figure in words

- According to Kerckhoffs' Principle, all parties know the algorithms $(KeyGen, F, I)$.
- The challenger $\mathcal{C}$ generates a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow KeyGen$ and picks at random a value $x \xleftarrow{\$} X$.
- The challenger $\mathcal{C}$ sends to the adversary $\mathcal{A}$ the public key pk and the value $y = F(\mathsf{pk}, x)$.
- Then the adversary $\mathcal{A}$ outputs x^*
- The adversary $\mathcal{A}$ wins iff $F(\mathsf{pk}, x^*) = y$.

The trapdoor permutation is said to be **secure** if the probability that the adversary $\mathcal{A}$ wins the game is negligible. Formally:

$$Pr\left[\begin{pmatrix} x^* \leftarrow \mathcal{A}(\mathsf{pk}, y) \\ \wedge F(\mathsf{pk}, x^*) = y \end{pmatrix} \middle| \begin{pmatrix} (\mathsf{pk}, \mathsf{sk}) \leftarrow KeyGen(1^n) \\ x \xleftarrow{\$} X, F(\mathsf{pk}, x) = y \end{pmatrix}\right] \leq negl(n)$$

In Definition 1, the key generation $KeyGen$ algorithm is an key generation algorithm that takes in input 1^n, specifying the secuirty level (length of the key), and outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$. The secret key sk is the trapdoor needed to compute the inverse algorithm. The forward algorithm, F, given as input the public key pk, allows to efficiently compute the output $y \in Y$ for every input $x \in X$. Remark that anyone can compute y given x, since this algorithm does not require knowledge of the private key. Finally, if we know the trapdoor sk, given y, we can efficiently compute the corresponding $x' = I(\mathsf{sk}, y)$.

Note: in general, x and x' could be different (there could be more preimage for the same value). However it always hold that $F(\mathsf{pk}, x) = y = F(\mathsf{pk}, x')$ is the same. Moreover, for RSA and permutations in general, it holds that $x = x'$ because they are one-to-one functions, thereby the preimage is unique.

Theorem 1. RSA is a trapdoor permutation: Let $N = p \cdot q$ (the product of two primes) be an integer and $e \in \mathbb{Z}^*_{\varphi(N)}$. The function $f_{N,e}(x) = x^e \mod N$ is a trapdoor permutation over $\mathbb{Z}^*_N$.

Proof. We need to show that RSA fits the previous definition. To do so, we define the three algorithms $(KeyGen, F, I)$ that make a trapdoor function in RSA:

$KeyGen(1^n) \to (\mathsf{pk} = (N, e), \mathsf{sk} = (p, q, d))$, where $N = p \cdot q$, $e \in \mathbb{Z}^*_{\varphi(N)}$,$d = e^{-1} \mod \varphi(N)$;

$F(\mathsf{pk}, x) = x^e \mod N$ is an efficient algorithm, it only involves exponentiation (efficiently computed using binExp), Euler's theorem, and modular arithmetic.

$I(\mathsf{sk}, y) = y^d \mod N$ is an efficient algorithm given the trapdoor d. Furthermore, we can see that I is an inverse for F from:

$$I(\mathsf{sk}, F(\mathsf{pk}, x)) = (x^e \mod N)^d \mod N = x^{ed} \mod N = x^{ed \mod \varphi(N)} \mod N = x^1 \mod N$$

Finally, we have that F is computationally hard to invert given only $\mathsf{pk} = (N, e)$ by the RSA assumption.

□

Trapdoor Permutations are important we can build the most secure notion of public key encryption using them (more on this later on in the course).

2 Digital Signatures

Think about a *handwritten* signature, it is something used to *prove that I signed the document* and ideally I should be the only one who can produce that specific signature, assuming now unique handwriting. The aim of digital signatures is to achieve the same digitally, we want to produce something that only the one with knowledge of the secret key could produce. Moreover, we want that everybody can be convinced about the validity of such a signature, meaning that it was indeed the claimed person who produced it.

2.1 Definition

A core difference between using symmetric cryptography and public key asymmetric cryptography is highlighted by the following example: in symmetric cryptography Alice and Bob know the same symmetric key k, thus cryptographically they are the same person. Thereby Alice cannot convince a third party that it was she, and not Bob, producing/texting something since they both have knowledge of the same key k. That is, a keyed MAC cannot be used by Alice to prove her identity to external parties. Since *whatever Alice produces, Bob can produce as well and vice versa!* This issue can be solved using asymmetric (public key) cryptography, as Alice is the only one who knows sk. If she uses it to do something that is (computationally) impossible to do without sk, then everyone can be convinced that she did it.

The public key equivalent of a MAC is called a **digital signature**. Digital signatures enable an entity to authenticate themselves to another entity, without any shared secret.

Definition 2. A **digital signature scheme** is a triple of PPT algorithms $(KeyGen, Sign, Ver)$ defined as follows:

- $KeyGen(1^n) \to (\mathsf{pk}, \mathsf{sk})$ is a probabilistic key generation algorithm. sk is secret key only known by signer and pk is the corresponding public key;
- $Sign(\mathsf{sk}, m) \to \sigma$ is a (possibly) probabilistic algorithm that outputs a valid signature σ for a message m;
- $Ver(\mathsf{pk}, m, \sigma) \in \{0, 1\}$ returns 1 if σ is accepted as a valid signature for m against pk, or 0 otherwise.

The digital signature scheme satisfies **correctness** if: for all key pairs $(\mathsf{pk}, \mathsf{sk}) \leftarrow KeyGen(1^n)$ and for all messages m, it holds that $Ver(pk, m, Sign(\mathsf{sk}, m)) = 1$.

The signing algorithm, $Sign$ requires the secret key sk as input, since only the owner of the secret key sk should be able to produce a valid signature. The correctness of a digital signature states that if the signature is produced in the right way, the verification algorithm Ver should accept the signature. A small remark, we write "*(possibly) probabilistic*" because better security comes with including randomness in the signature generation process, however, a very iconic signature scheme (textbook RSA, that we will present momentarily) is deterministic.

2.2 Towards Defining Security for Digital Signatures

To define the security notion of a digital signature one has to take two aspects into account: the adversary's power, its knowledge and its goal(s).

Adversary's power and knowledge

We consider three different adversaries that have different power and knowledge.

- **Key-Only Attack:** $\mathcal{A}$ knows only the signer's pk, and therefore only has the capability of checking the validity of signatures of messages. For example, we can think about this attack as if we shared a public key on the web without signing any message. What could the adversary do? Remark that $\mathcal{A}$ can neither see nor verify signatures.
- **Known Signature Attack:** $\mathcal{A}$ knows pk and sees message/signature pairs chosen and produced by the legal signer. We can think about this attack as if we share a public key on the web and use it to sign, e.g., emails. The adversary may send an email to the signer, and the reply will contain a signature for a message of the signer's choice. In this way, the adversary is going to see the signatures for some messages, but it still lacks power to choose which messages should be signed.
- **Chosen Message Attack:** $\mathcal{A}$ knows pk and can ask the signer to sign several messages of the adversary's choice. This attack could seem unrealistic, but for example, we can download some software with a bug that forces us to sign a specific sequence of messages. It could seem to be a really strong power, but it happens in the real world more often than we think.

In order to formally model the fact that the adversary $\mathcal{A}$ can request signatures of specific messages by giving $\mathcal{A}$ access to a *signing oracle*. The signing oracle, $\mathcal{O}^{Sign}_{\mathsf{sk}}$, works as follos: given a message as input, the oracle outputs a valid signature for that message signed using the secret key sk corresponding to the public key pk that was given to the adversary.

Adversary's goal

We consider four different goals of the adversary.

- **Existential Forgery:** $\mathcal{A}$ succeeds in creating a valid signature of a new message (a message for which $\mathcal{A}$ has not seen a valid signature).
- **Strong Forgery:** $\mathcal{A}$ succeeds in creating a valid signature of some message of $\mathcal{A}$'s choice **and** the signature is different from any signature seen by $\mathcal{A}$.
- **Universal Forgery:** $\mathcal{A}$ is able to generate a valid signature for any message of his choice (remark $\mathcal{A}$ does not know sk).
- **Total Break:** $\mathcal{A}$ can compute the signer's secret key sk.

The **Strong Forgery** is a powerful notion, it means that the adversary $\mathcal{A}$ can choose which messages it would like to sign. The main difference between a *Strong Forgery* and *Universal Forgery* is the in the former the adversary $\mathcal{A}$ has the power to built a signature for *some* messages (e.g., only special combinations of messages/signatures it received), in the latter, $\mathcal{A}$ has the power to generate a signature for *any* (and usually many) messages.

The recipe for a good security notion is to choose a realistic adversary (PPT, Quantum...), give $\mathcal{A}$ the strongest starting knowledge (i.e. Chosen Message Attack) and select the weakest damage to the cryptosystem (adversary's goal above). Therefore in the case of digital signatures we consider: **Existential Forgery under Chosen Message Attack, (EUF-CMA)**.

2.2.1 EUF-CMA: The Existential Forgery under Chosen Message Attack Security Game

We now look closer into the definition of EUF-CMA. Let us create a security game for digital signatures with the aim of quantifying $\mathcal{A}$'s likelihood in creating a valid (EUF-CMA) forgery of a signature σ^* for a **new** message m^*. Such a game is seen in Figure 1

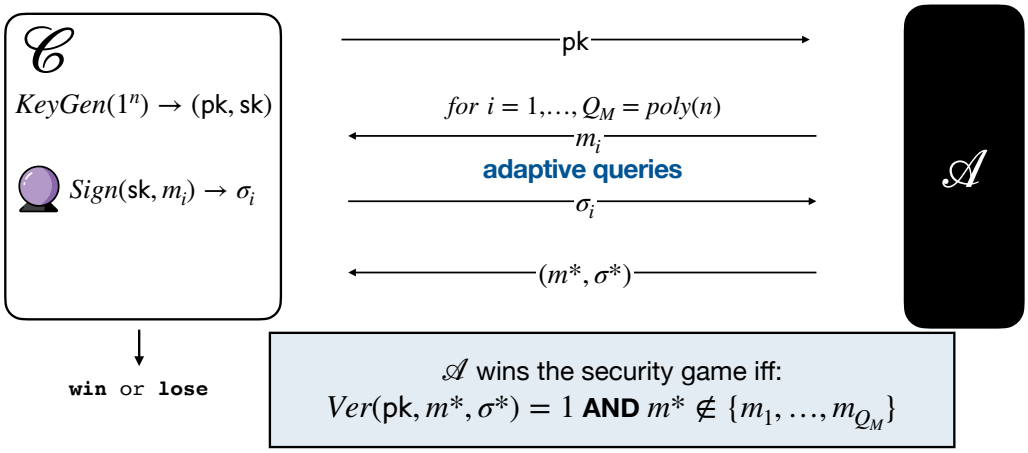


Figure 1: EUF-CMA security game

Describing the above figure in words

- All the parties know the algorithms according to Kerckhoffs' principle.
- The challenger $\mathcal{C}$ starts by generating a key pair $KeyGen(1^n) \rightarrow (\mathsf{pk}, \mathsf{sk})$.
- The challenger $\mathcal{C}$ then sends pk to the adversary $\mathcal{A}$.
- The adversary $\mathcal{A}$ can then send polynomially many queries ($i \in Q_m (= poly(n))$) of messages m_i to the signing oracle (hosted by the challenger $\mathcal{C}$), who will respond with the corresponding signatures σ_i.
- Then $\mathcal{A}$ outputs a new message-signature pair (m^*, σ^*).
- The adversary $\mathcal{A}$ wins the security game if and only if $Ver(\mathsf{pk}, m^*, \sigma^*) = 1$ **and** $m^* \notin \{m_1, ..., m_{Q_m}\}$, i.e. the adversary $\mathcal{A}$ wins if he can produce a pair (m^*, σ^*) that verifies and the message is not one of the queried messages, it needs to be a new message.

The adversary $\mathcal{A}$ need to send the pair (m^*, σ^*) because if he send only the signature, it is not possible to check the validity of the signature, we need also the message. Moreover, we ask for a signature for a new message because the adversary has already seen valid signature on those messages.

Definition 3. A digital signature scheme is said to be **secure** (EUF-CMA) if for all efficient (PPT) adversaries the probability that the adversary $\mathcal{A}$ wins the EUF-CMA security game is negligible. Formally,

$$Pr[Ver(\mathsf{pk}, m^*, \sigma^*) = 1 | (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}^{Sign}}(\mathsf{pk}) \wedge m^* \notin \{m_i\}_{i=1}^{Q_M}] \leq negl(n),$$

2.3 The RSA Signature Scheme

As an example of a digital signature scheme we consider the Textbook RSA signature. The term *Textbook* refers to the version that is most often introduced in class, which is a simplification of what is actually used in practice.

Textbook **RSA Signature Scheme** is defined by the three PPT algorithms $(KeyGen, Sign, Ver)$:

- $KeyGen(1^n) \rightarrow (\mathsf{pk}, \mathsf{sk})$: pick two $n-$bit long primes p, q and compute $N = p \cdot q$. Then pick a random invertible element $e \xleftarrow{\$} \mathbb{Z}^*_{\varphi(N)}$, and compute its inverse $d \mod \varphi(N)$. *Remark that $\varphi(N)$ can be efficiently computed given the knowledge of p, q.* Finaly set $\mathsf{pk} = (N, e)$ and $\mathsf{sk} = (p, q, d)$.
- $Sign(\mathsf{sk}, m) \rightarrow \sigma$: given $m \in \mathbb{Z}^*_N$, return $\sigma = m^d \mod N$;
- $Ver(\mathsf{pk}, m, \sigma) \in \{0, 1\}$: returns 1 iff $m = \sigma^e \mod N$.

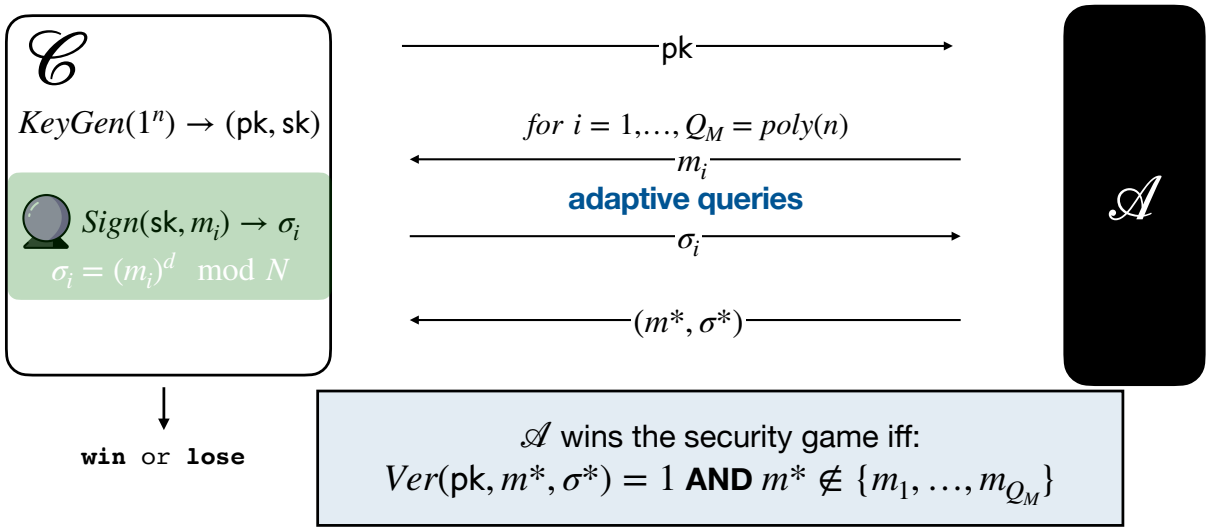
To show that this scheme satisfies **correctness** we consider the verification algorithm Ver, to check that it works as expected.

$$\sigma^e \mod N = (m^d)^e \mod N = m^{d \cdot e} \mod N = m^{1 \mod \varphi(N)} \mod N = m \mod N$$

The **security** of the scheme relies on the RSA assumption: given the public key, is hard to find the exponent d. For good security use $n = 1024$ (hence $\log_2(N) = 2048$). To have an intuition, a number of 2048 bits is long in decimal form in this font size takes 8.5 meters.

Is the textbook RSA signature EUF-CMA secure?

To check this we need to quantify the $\mathcal{A}$'s likelihood in forging a valid signature σ^* for a new message m^*. i.e. output a valid EUF-CMA forgery.



The RSA Signature Scheme is not EUF-CMA secure, which is shown by the following atack:

$$\mathsf{pk} = (N, e)$$
$$m_1 \leftarrow \$\mathbb{Z}_N^* \backslash \{1\}$$
$$\sigma_1 \leftarrow \mathcal{O}_{sk}^{Sign}(m_1)$$
$$m_2 \leftarrow \$\mathbb{Z}_N^* \backslash \{1\}$$
$$\sigma_2 \leftarrow \mathcal{O}_{sk}^{Sign}(m_2)$$

$$m^* = m_1 \cdot m_2 \mod N$$
$$\sigma^* = \sigma_1 \cdot \sigma_2 \mod N$$

- The adversary $\mathcal{A}$ can query two different messages m_1 and m_2. The oracle $\mathcal{O}_{\mathsf{sk}}^{Sign}$ replies with two valid signatures σ_1 and σ_2.
- Now the adversary $\mathcal{A}$ has the power to build a valid signature for a message m^* that it has never queried before. Simply by setting the message $m^* = m_1 \cdot m_2 \mod N$ and the corresponding signature $\sigma^* = \sigma_1 \cdot \sigma_2 \mod N$.

We see by the below computations that (m^*, σ^*) is a valid messsage-signature pair.

$$Ver(\mathsf{pk}, m^*, \sigma^*) = 1$$
$$\Updownarrow$$
$$(\sigma^*)^d = (\sigma_1 \cdot \sigma_2)^d = \sigma_1^d \cdot \sigma_2^d = m_1 \cdot m_2 = m^* \mod N$$

Why do we use RSA Signatures if they are not EUF-CMA? Because it is not the textbook version of the cryptosystem that is implemented in practice! For EUF-CMA signing use RSA with Full Domain Hash Signing (RSA-FDHS)!

2.4 The Hash-and-Sign Paradigm

Now we describe a generic way to build signatures that are unforgeable under a chosen message attack (i.e. EUF-CMA secure) by using a hash function and applying a trapdoor permutation. That is we will use Trapdoor Permutations in combination with collision-resistant hash functions to make secure digital signatures. An example of this is the Hash and Sign Paradigm (aka Full Domain Hash Signatures = FDHS).

Example 1. FDHS Recipe: Given a trapdoor one-way function $(KeyGen, F, I)$ over X, Y and a cryptographic hash function $H : \{0,1\}^* \to X$ build a digital signature scheme as follows:

- $KeyGen_{Sig}(1^n) = KeyGen_{TD}(1^n) \to (\mathsf{pk}, \mathsf{sk})$, hence the $KeyGen$ algorithm of the signature scheme is exactly the $KeyGen$ algorithm of the trapdoor function.
- $Sign(\mathsf{sk}, m)$: compute $\sigma = I(\mathsf{sk}, H(m))$
- $Ver(\mathsf{pk}, m, \sigma)$: if $F(\mathsf{pk}, \sigma) = H(m)$ return 1, else 0.

Theorem 2. If $(KeyGen_{TD}, F, I)$ is a one-way trapdoor function and H is a random oracle, then FDHS is unforgeable (i.e., FDHS is a chosen message attack secure digital signature scheme).

Proof. The proof is not a part of the course, if you are interested, see Theorem 10.17 in https://intensecrypto.org/public/lec_11_concrete_pkc.html#hardcore-bits-and-security-without-random-oracles

□

Some reamarks about RSA-FHDS:

- We cannot use $sha256$ as a hash function for the RSA signature because this function reduces too much the domain space.
- RSA provides a very versatile trapdoor function: swapping the roles of F and I we get a digital signature scheme!

Intuition of homomorphic signatures

We have seen that RSA is not EUF-CMA secure since we could multiply two signatures and generate a new signature of the multiplication of the two messages signed by the original signatures.

Sometimes, being able to combine different signatures gives rise to the possibility of efficiently verifying the correctness of computations on authenticated data that is outsourced to a cloud server. Such signatures are called **homomorphic**, a term borrowed from mathematics that refers to "transformations of one data set into another while preserving relationships between elements in both sets".

For example, consider the case of a data analyst collecting pollution data and uploading it to the cloud. Subsequently, statisticians may query the database to compute the average air quality for a specific month, obtaining a numerical result. How can we ensure that this result is computed correctly, using all the relevant data for the specified month? Requesting access to all the raw data might appear to be the simplest solution; however, this is not always feasible due to potential privacy concerns or the impracticality of downloading large datasets. Homomorphic signatures provide a solution to this problem. The data analyst in the example above, can now upload authenticated values in the database (i.e., message-signature pairs). Since the signature scheme employed is homomorphic, given a function f (e.g., the average of the pollution values during a specific month) the cloud can compute on the signatures to obtain one valid signature that authenticates the average pollution value for the queried month. Homomorphic signatures have additional machinery that ensures the correct data (and its signatures) was used by the cloud when computing the answer. This situation is schematized in the Figure 2.

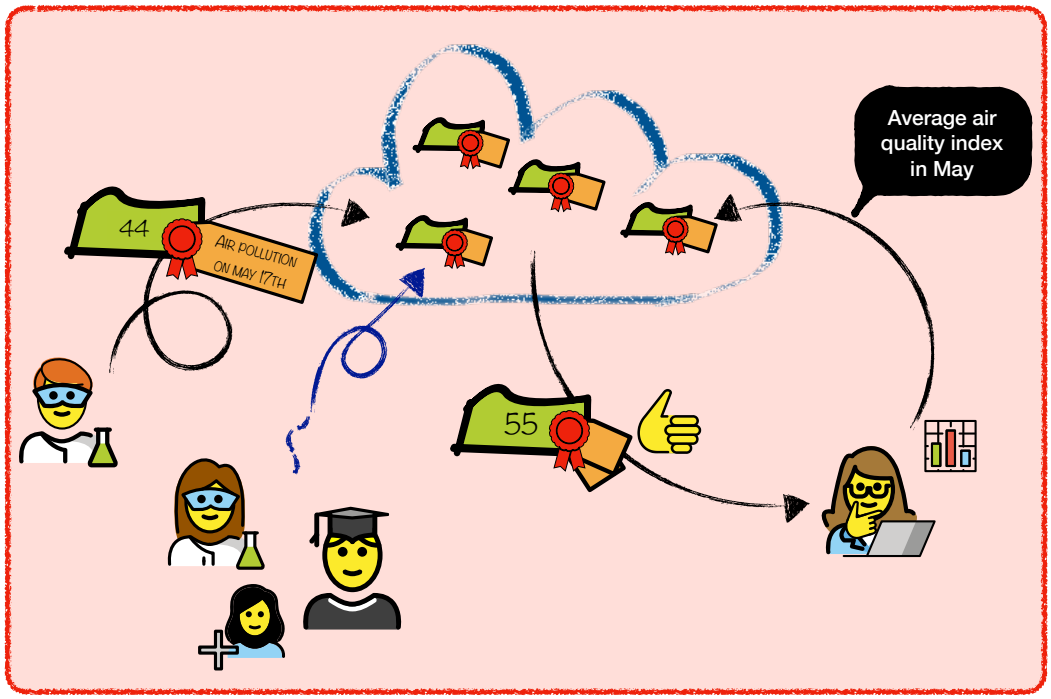


Figure 2: Illustrative example of Linear Homomorphic Signatures

2.5 Elliptic Curve Digital Signature Algorithm (ECDSA) (Not in Exam)

Elliptic curves are created by functions for the form $y^2 = x^3 + ax + b$, see Figure 3 for a visualization of such a curve. Elliptic curves have a group structure[1], and can therefore be used (similarly to other groups we have seen such as $Z_N, Z_p^*, Z_n^*, \mathbb{G}$) to build cryptographic primitives such as in e.g. digital signature schemes.

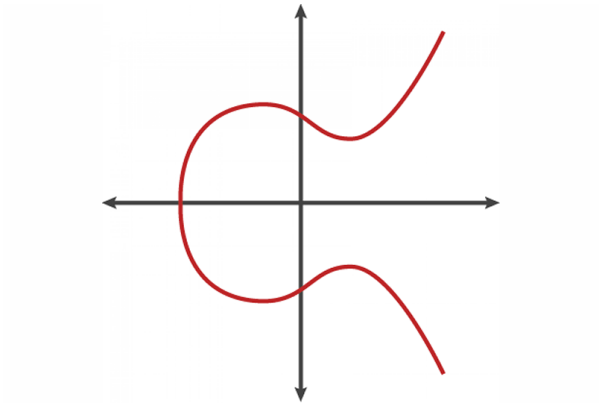


Figure 3: Graph of a function $y^2 = x^3 + ax + b$.

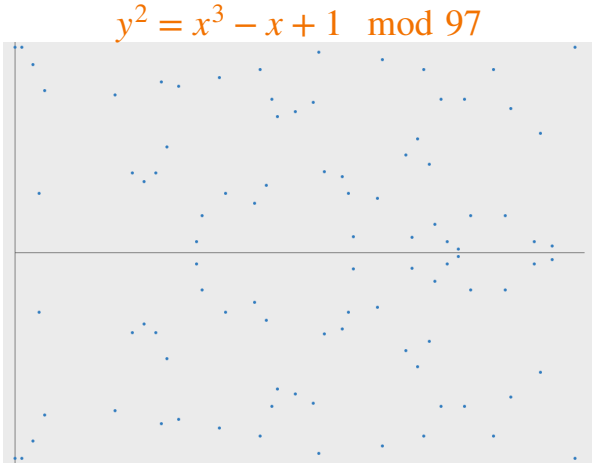


Figure 4: Points of the group of a function $y^2 = x^3 - x + 1 \mod 97$.

```
KeyGen(sec.par) -> (sk, pk):
    d <- $[0...n-1]
    sk = d
    pk = Q = d*G
```

Listing 1: Method for generating keys using ECDSA

```
Sign (sk, msg) -> sgn
    k <- $[0...n-1]
    R = k*G
    r = R_x mod n
    z = sha256(msg)
    s = inv(k)*(z+d*r) mod n
    sgn = (r, s)
```

Listing 2: Method for signing a message using ECDSA

The point R is a random point on the elliptic curve since it is computed as the generator G multiplied by a random number k.

```
Veryfiy(pk, msg, sgn) -> {0,1}
 z = sha256(msg)
 T = [z*inv(s) mod n] * G
 P = [inf(s)*r mod n] * Q
 if R == t+p return 1
 else return 0
```

Listing 3: Method for verifying signatures using ECDSA

Listings 1-3 present three methods for implementing a digital signature scheme using elliptic curves, specifically **ECDSA**(Eliptic Curve Digital Signature Algorithm). The advantages of ECDSA are that it has shorter keys and better security than the RSA signature scheme, it is non malleable and it is IoT friendly. These advantages makes it a well used signature scheme in many realworld applications.

Digital Signatures are used daily, this is an example of the certificate issued by the webpage GitHub.

```
    0 *.github.io
[...]
Issuer Name
    Country or Region US
    Organisation  DigiCert Inc
    Common Name DigiCert Global G2 TLS RSA SHA256 2020 CA1
[...]
Public Key Info
    Algorithm RSA Encryption ( 1.2.840.113549.1.1.1 )
    Parameters  None
```

[1] see JupyterNotebooks file Elliptic.ipynb

```
    Public Key  256 bytes: AD 2B 14 A5 3A 4C 41 AF [...]
    Exponent  65537
    Key Size  2 048 bits
    Key Usage Encrypt, Verify, Wrap, Derive
[...]
   Signature Algorithm  SHA-256 ECDSA
[...]
Fingerprints
    SHA-256 09 01 0C CE 9B 72 21 55 C7 E6 86 B77 39 D3 D2 [...]
    SHA-1           97 D8 C5 70 0F 12 24 6C 88 BC FA 06 7E 8C A7 4D [...]
```

Listing 4: Part of the certificate for https://epagnin.github.io/

The Public Key information uses RSA Encryption scheme, while the Signature algorithm used is SHA-256 ECDSA signature scheme.

Examples of faulty realworld usage of ECDSA

The usage of ECDSA has not always been perfect in the real world, but this is because of faulty implementations, not the secuirty of the cryptographic signature scheme. For example, the PS3 in 2010 used ECDSA for digital signatures, but it did not create new nonces but rather reused old ones. Adversaries could therefore retrieve the secret key after intercepting two different messages with signatures. This implies that some users were able to play games for free. This attack happened again in 2013 when the Android bug batters Bitcoin wallets. There are also more recent witnesses of bad implementations, as in 2020 with LadderLeak.

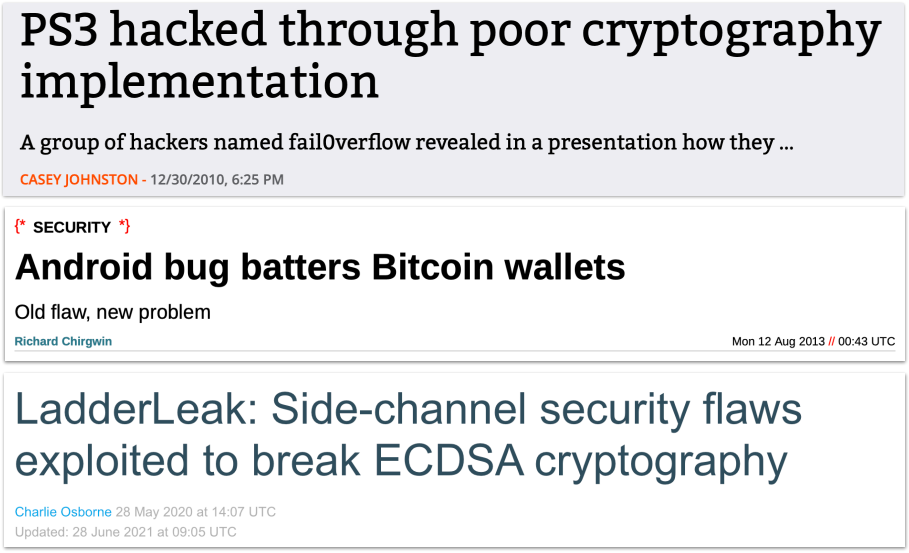


Figure 5: Witnesses of bad implementations using ECDSA

TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Public key encryption, Homomorphic encryption

Lecturer: Ivan Oleynikov Lecture 6 on Nov 22

Scribes: Willem Brahmstaedt, Hanna Ek & Lucia Lavagnino Last update December 17, 2024

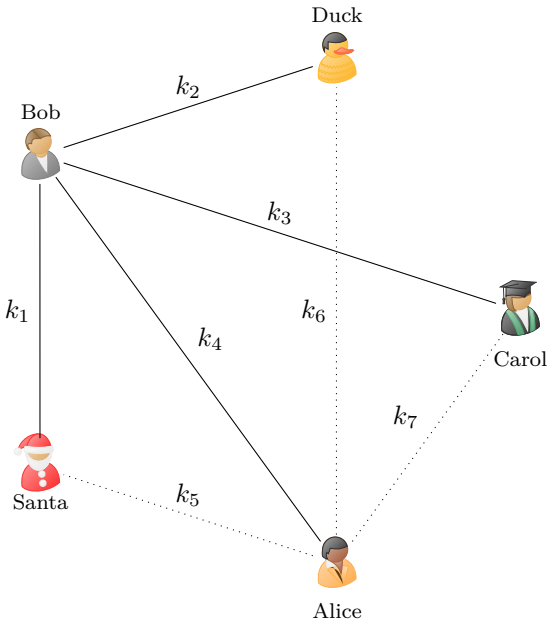
Contents

1 Public Key Encryption (PKE)

Motivation for PKE

Imagine n people who want to establish confidential channels between each pair of people. How many symmetric encryption keys would be needed for that?

- Referring to the figure, to connect all the people we need 10 keys. 4 keys for securing communication with Bob, additional 3 keys for communicating to Alice, 2 more keys for connecting Carol (to Duck and Santa), and 1 key to create the final link between Duck and Santa.
- For the general case of n people: $\frac{n(n-1)}{2}$ keys.



Since every connection need its own key, we needs $\binom{n}{2} = \frac{n(n-1)}{2}$ keys in total. This number grows fast with the number of users. Public key encryption, which we will study in this chapter, reduces the number of keys needed to establish secure communication drastically as we will see next. With public key encryption, each user can generate a pair of correlated keys: public key and secret key. Public key encrypt messages, and the corresponding secret key decrypts them.

In the example with n users which we presented above, each user needs to generate its own key pair, and only needs to distribute the public key.

For example, Alice creates a keypair and distributes her public key to all other users, i.e., each of the remaining $n-1$ users. Then all of them can send confidential messages to Alice by using Alice's public key to encrypt and Alice can decrypt by using her secret key.

Thus with a PKE scheme, each of the n users will need to generate one key-pair, thereby n key pairs in total, compared to the $\binom{n}{2} = \frac{n(n-1)}{2}$ keys needed in case of symmetric encryption.

Definition 1. A **Public-Key Encryption (PKE)** is a triple of PPT algorithms $(KeyGen, Enc, Dec)$ defined as follows:

- $KeyGen(1^n) \to (\mathsf{pk}, \mathsf{sk})$ is a probabilistic key generation algorithm, it takes in input the security parameter (length of the secret key), and it outputs a public key and a secret key.
- $Enc(\mathsf{pk}, m) \to c$ is a (possibly) randomised encryption algorithm, it takes in input a public key and a message (plaintext), and it returns a ciphertext.
- $Dec(\mathsf{sk}, c) \to m'$ is a deterministic decryption algorithm; it takes in input a secret key and a ciphertext, and it returns a message.

Correctness: A PKE scheme is said to be correct if for any pair of keys generated by $KeyGen$, for any message m it holds that: $m = Dec(\mathsf{sk}, Enc(\mathsf{pk}, m))$.

The decryption algorithm Dec is deterministic, i.e. on the same input it always gives the same output. The $KeyGen$ and (usually) Enc are randomised, which means that on the same input, they may produce different outputs. This property is fundamental for key generation, and optional for encryption (though randomness improves security, as we will see). The correctness property states that the decryption will always return the original plaintext, if the key generation and the encryption algorithm as run as expected.
The definition of PKE is similar to the one of One-Way Trapdoor Functions (TDF) although there is a core difference: in a TDF built by $(KeyGen, F, I)$ the correctness relies on the evaluation of the function, i.e. if $x' = I(\mathsf{sk}, F(\mathsf{pk}, x))$, we ask that $F(\mathsf{pk}, x) = F(\mathsf{pk}, x')$ but there is no guaranteed that $x = x'$, we are only sure that x and x' map on the same value through F. On the other side, in PKE, correctness means that the plaintext linked to a specific ciphertext is unique, i.e. $m = Dec(\mathsf{sk}, Enc(\mathsf{pk}, m))$. Moreover, the input of a TDF x is chosen at random, while in PKE is a specific message m we would like to transmit.

Remarks of One-way Trapdoor Functions

- One-Way Trapdoor Functions are equivalent to (IND-CCA) Public-Key Encryption, in the sense that we can always build PKE from TDFs, and viceversa.
- TDFs can also be used to build signature schemes, if combined with a collision resistant hash function.

1.1 Textbook RSA

The first public key cryptography scheme we will introduce is Textbook RSA. It consists of three algorithms, $KeyGen$, Enc and Dec, described below.

KeyGen(1^n)

1: Pick two n-bit long primes p and q
2: $N = p \cdot q$
3: Pick $e \xleftarrow{\$} \mathbb{Z}^*_{\varphi(N)}$
4: compute $d = e^{-1} \mod \varphi(N)$
5: $\mathsf{pk} = (N, e)$
6: $\mathsf{sk} = (N, d)$
7: **return** $(\mathsf{pk}, \mathsf{sk})$

Enc($\mathsf{pk}, m \in \mathbb{Z}^*_N$)

1: $c = m^e \mod N$
2: **return** c

Dec($\mathsf{sk}, c \in \mathbb{Z}^*_N$)

1: $m = c^d \mod N$
2: **return** m

Remarks

- Picking $e \in \mathbb{Z}^*_{\varphi(N)}$ is equivalent to picking $e < N$ such that $\gcd(e, \varphi(N)) = 1$.
- For efficiency, use Binary Exponentiation to raise values to powers d and e modulo N.

- The inverse of e modulo $\varphi(N)$ can be efficiently computed using the *Extended Euclidean Algorithm* or *Euler's theorem.*

1.1.1 Correctness

Next, we show that textbook RSA is a correct PKE scheme, in other words that the decryption of an encryption of m will give the same message m back. Let p and q be two distinct primes, $N = pq$, $d, e \in \mathbb{Z}^*_{\varphi(N)}$ such that $d = e^{-1} \mod \varphi(N)$. Then for any $m \in \mathbb{Z}^*_N$, it holds that $(m^e)^d = m \mod N$

From the definition of textbook-RSA we have that $d = e^{-1} \mod \varphi(N)$, and from this we can conclude that $e \cdot d = 1 \mod \varphi(N)$. This means that there exists such k that $d \cdot e = 1 + k \cdot \varphi(N)$. Now let us replace $e \cdot d$ by this expression in $m^{e \cdot d}$: $m^{e \cdot d} = m^{1+k\varphi(N)} = m \cdot m^{k\varphi(N)} = m \cdot (m^{\varphi(N)})^k$. By Euler's Theorem, $m^{\varphi(N)} = 1 \mod N$. Therefore,

$$m^{e \cdot d} = m \cdot (m^{\varphi(N)})^k = m \cdot 1^k = m \mod N.$$

1.2 ElGamal

Now we introduce another PKE scheme, namely ElGamal. As we will see there are similarities between ElGamal and the Diffie-Hellman key-exchange. Therefore to describe ElGamal PKE we start with DH key exchange and then expand it into ElGamal.

We first recall that the Diffie-Hellman key exchange enables two parties to share a secret key between themselves while communicating via an insecure channel and without relying on any pre-existing secret shared between the two parties.

Let be p a large prime and g an element in $\mathbb{Z}^*_p$ which generates a cyclic (multiplicative) subgroup of prime order q. Given this notation, in Figure 1 (above the horizontal line) we present a protocol where Alice and Bob run Diffie–Hellman key exchange and (below the horizontal line) Alice uses the generated key to one-time-pad encrypt a message m and send the encryption to Bob, who receives and decrypts that message using the shared DH-key.

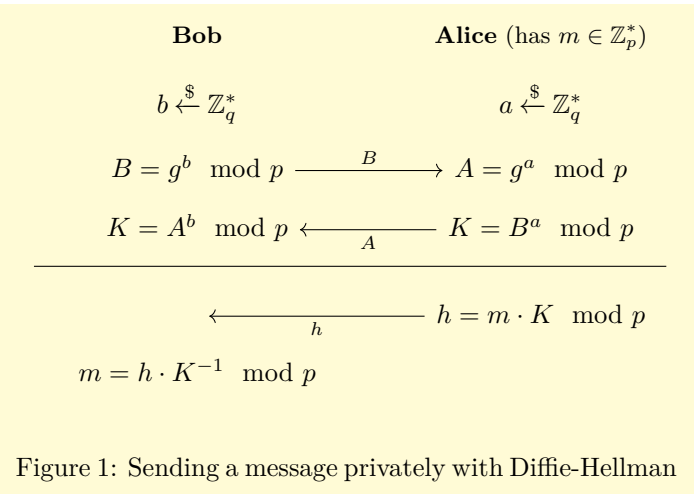


Figure 1: Sending a message privately with Diffie-Hellman

1. Bob and Alice sample at random values b resp. a from $\mathbb{Z}^*_q$. b is *Bob*'s private key while a is *Alice*'s private key;
2. Bob computes his public key as $B = g^a \mod p$ and shared the public key with Alice;
3. Alice also computes her public key as $A = g^b \mod p$ and shared the public key with Bob;
4. then, both Alice and Bob can compute a shared key K as $A^b = K = B^a$.
5. Finally, Alice can send the encryption h of the message m to Bob using OTP through the shared key K.

This protocol keeps the message secret from someone who is observing the messages being sent between Alice and Bob. Assuming DH is secure, no party can learn K by a passively observing adversary. Hence, no observer (adversary) can learn$m$ from h.

We can decompose the protocol above into a public key encryption scheme, by framing the steps that identify each of the three algorithms as illustaed by the blue boxes present in Figure 2.

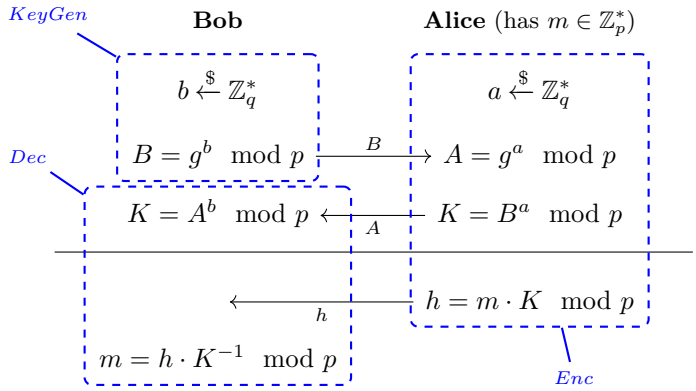


Figure 2: Diffie–Hellman used as a public key encryption scheme

In this case, b would be Bob's secret key, and B his public key. Alice can then encrypt a message to Bob by computing a shared key $K = B^a$ and sending the cipher text $(A = g^a, c = m \cdot K)$ to Bob. Bob can then decrypt the message by first computing the shared key $K = A^b$, and then using its inverse compute the message $m = c \cdot K^{-1}$. The scheme obtained this way is called ElGamal.

A formal and precise definition of ElGamal's three algorithms is defined below:

$KeyGen(1^n)$

1 : Pick an n-bit long prime p
2 : Pick an element $g \in \mathbb{Z}_p^*$ that generates a subgroup of prime order q
3 : $b \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$
4 : $B = g^b \mod p$
5 : $\mathsf{sk} = b$
6 : $\mathsf{pk} = B$
7 : **return** $(\mathsf{pk}, \mathsf{sk})$

$Enc(\mathsf{pk}, m \in \mathbb{Z}_p^*)$

1 : $B = \mathsf{pk}$
2 : $a \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$
3 : $A = g^a \mod p$
4 : $K = B^a \mod p$
5 : $h = K \cdot m \mod p$
6 : $c = (A, h)$
7 : **return** c

$Dec(\mathsf{sk}, c \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*)$

1 : $b = \mathsf{sk}$
2 : $(A, h) = c$
3 : $K = A^b \mod p$
4 : $m = h \cdot K^{-1} \mod p$
5 : **return** m

Figure 3: ElGamal encryption scheme

ElGamal works with the following two parameters:

- Prime modulus p and the corresponding group $\mathbb{Z}_p^*$.
- Generator $g \in \mathbb{Z}_p^*$ that generates a subgroup $\mathbb{G} = \langle g \rangle$ of prime order q.

We do not specify how the parameters p, g, q are picked in our KeyGen algorithm, because ElGamal does not rely on a specific way of doing it. As long as the given g generates a cyclic group of order q, ElGamal will work correctly. To ensure security, q must be large enough so that brute-force attacks are infeasible (usually $q > 2^{1024}$).

One way to construct good p, g and q is to pick p to be a safe prime[1], so that $p = 2q + 1$. In this case, $|\mathbb{Z}_p^*| = \varphi(p) = 2q$. By Lagrange Theorem[2], the order any subgroup of $\mathbb{Z}_p^*$ must divide $2q$. In other words, the order of any subgroup G of $\mathbb{Z}_p^*$ can be only one of the values $1, 2, q, 2q$.

Knowing this, we can efficiently check if a given $g \in \mathbb{Z}_p^*$ (say, picked randomly) generates a subgroup of order q. We just check that $g^2 \neq 1$ and $g^q = 1 \mod p$. (And also ensure that $g \neq e$ rules out the subgroup of order 1.) Alternatively, one can use one of the predefined elliptic curve groups. Each such group comes with a known order q.

1.2.1 Correctness

Next we check that ElGamal satisfies correctness, that is, we verify that a message encrypted with ElGamal decrypts to the same message. To do so, consider $(KeyGen, Enc, Dec)$ as defined in Figure 3. We want to show that for any pk and sk output by $KeyGen(1^n)$ and for any message $m \in \langle g \rangle$, the following holds: $Dec(sk, (Enc(\mathsf{pk}, m)) = m$.

[1] https://en.wikipedia.org/wiki/Safe_and_Sophie_Germain_primes
[2] https://en.wikipedia.org/wiki/Lagrange%27s_theorem_(group_theory)

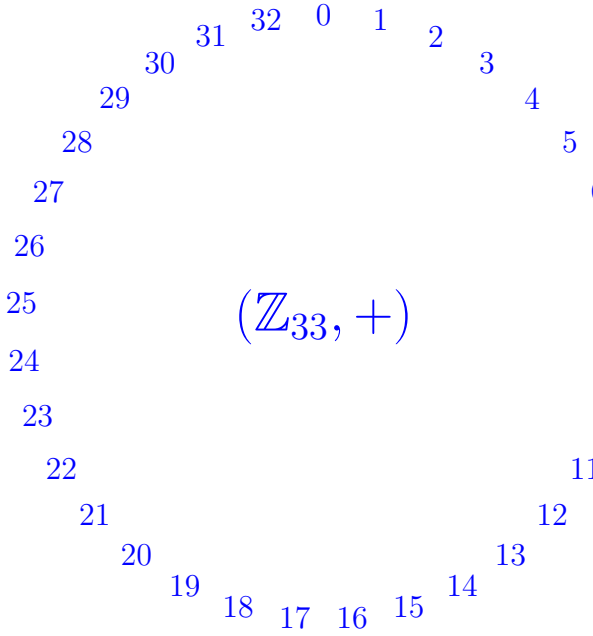
Observe that the K value used in *Enc*and in *Dec*the same. This is due to $A^b = B^a = g^{ab} \mod p$. Then $c \cdot K^{-1} = (m \cdot K) \cdot K^{-1} = m \mod p$.

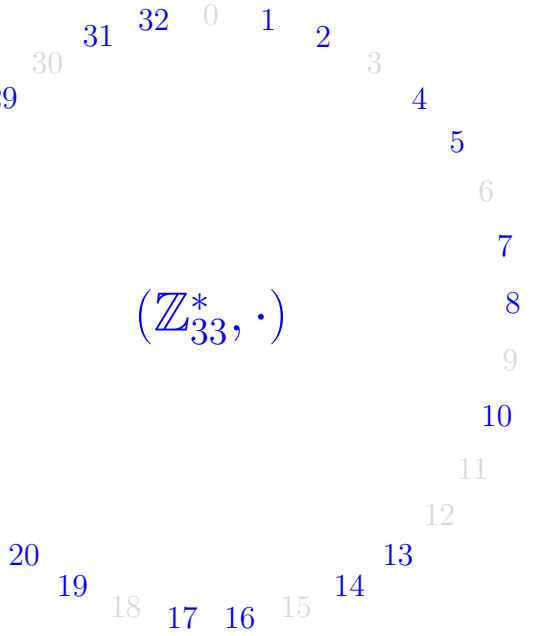
1.3 Group Theory for RSA and ElGamal

The observant reader will have noticed that RSA and ElGamal consider different groups. For example RSA is defined over the group $\mathbb{Z}_N^*$, where N is a composite number while in ElGamal we consider a subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ where p is a prime and $\mathbb{G}$ has prime-order q. This section aims to clarify why different groups are considered for RSA and ElGamal.

1.3.1 RSA encryption scheme

We consider an instance of RSA where the public and secret keys are $\mathsf{pk} = (33, e)$ and $\mathsf{sk} = (33 = 3 \cdot 11, d)$. Let us have a brief overview of the mathematical object we are going to work with:



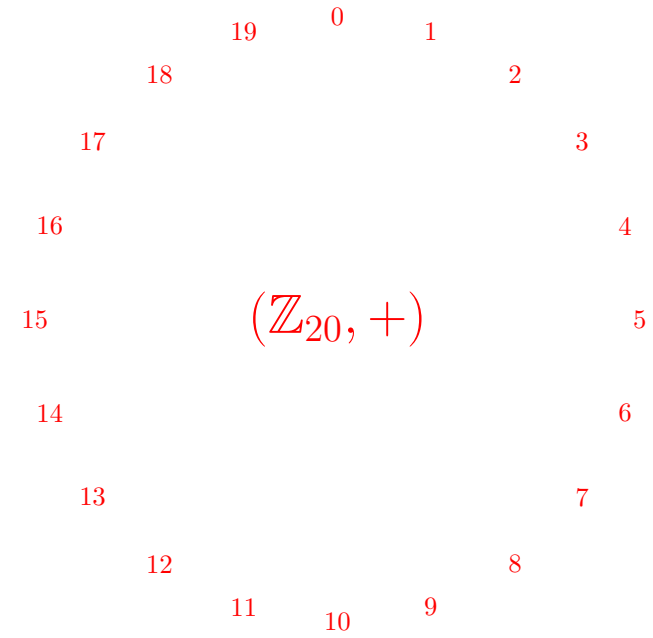


$|\mathbb{Z}_{33}| = 33$

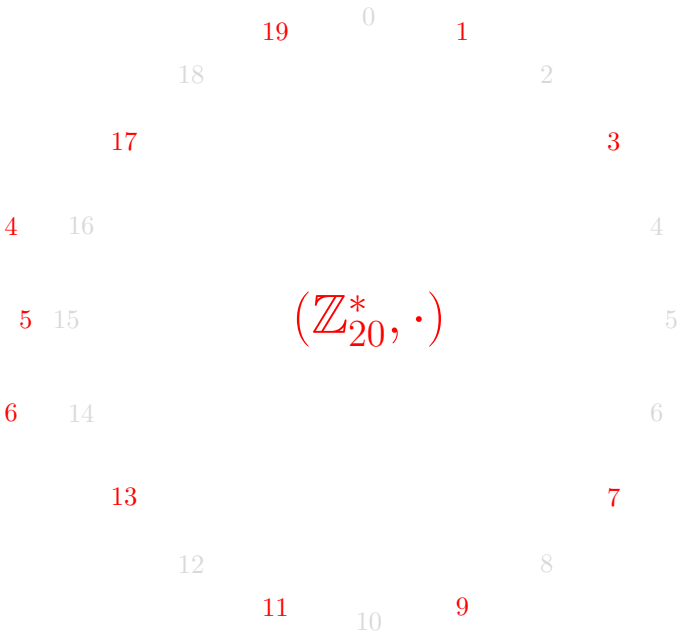
$|\mathbb{Z}_{33}^*| = \varphi(33) = \varphi(3) \cdot \varphi(11) = 2 \cdot 10 = 20$

- The additive group $(\mathbb{Z}_{33}, +)$ contains all the integer numbers $\{0, 1, 2, 3, \ldots, 32\}$.
- The multiplicative group $(\mathbb{Z}_{33}^*, \cdot)$ contains all the integer numbers smaller than 33 and coprime with 33, i.e. such all the numbers $z \in \mathbb{Z}_{33}$ that $\gcd(z, 33) = 1$.
- The Euler's Totient function tells us how many numbers are coprime with 33, hence how many numbers are in the group $(\mathbb{Z}_{33}^*, \cdot)$.
- Recall Euler' Theorem Collorary: $\forall a \in \mathbb{Z}_N^*, \forall x \in \mathbb{N}$, it holds that $a^x \mod N = a^{x \mod \varphi(N)} \mod N$.
  - Thus when doing exponentiations, we care about the exponent modulo $\varphi(N)$
  - And only consider exponents from the group $\mathbb{Z}_{\varphi(N)}$.

Since $\varphi(33) = 20$, in the textbook RSA, the exponents are from the group $\mathbb{Z}_{20}$. Moreover, to have an trapdoor one way function, we need the exponent e (the public key), to be invertible in $\mathbb{Z}_{\varphi(N)}$.



$|\mathbb{Z}_{20}| = 20$



$|\mathbb{Z}_{20}{}^*| = \varphi(2^2) \cdot \varphi(5) = (2-1) \cdot 2 \cdot 4 = 8$

What happens in RSA if $\mathbf{m} \notin \mathbb{Z}_{\mathbf{N}}^*$?
To illustrate this we pick as two messages, the numbers $m_1 = 12$ and $m_2 = 22$, that are in $\mathbb{Z}_{33}$, but **not** in $\mathbb{Z}_{33}^*$. We will see that this could lead to that we lose secrecy. To see why, consider the powers of 12 and 22 modulo 33:

$$\begin{array}{llll} 12^1 = 12 & \mod 33 & 22^1 = 22 & \mod 33 \\ 12^2 = 12 & \mod 33 & 22^2 = 22 & \mod 33 \\ 12^3 = 12 & \mod 33 & 22^3 = 22 & \mod 33 \\ 12^4 = 12 & \mod 33 & 22^4 = 22 & \mod 33 \\ 12^5 = 12 & \mod 33 & 22^5 = 22 & \mod 33 \\ \vdots & & \vdots & \end{array}$$

We can observe that for every possible choice of the public key e, there is no way to obtain a ciphertext that is different from the plaintext, hence there is no secuirty obtained by the encryption of the messages in $\mathbb{Z}_{33} \setminus \mathbb{Z}_{33}^*$.

What happens in RSA if $\mathbf{e} \notin \mathbb{Z}_{\varphi(\mathbf{N})}^*$?
Consider the message $m = 7 \in \mathbb{Z}_{33}^*$, and choose $e = 10$, note that $e \in \mathbb{Z}_{\varphi(N)}$ but $e \notin \mathbb{Z}_{\varphi(N)}^*$, consequently this is not an invertible element in $\mathbb{Z}_{\varphi(N)}^* = \mathbb{Z}_{20}^*$. We will show that this causes that the RSA cryptosystem is no longer correct, indeed encrypting using textbook RSA gives:

$$c = Enc(\mathsf{pk}, m) = m^e \mod N = 7^{10} \mod 33 = 1.$$

You can check that 7 generates a multiplicative group of order 10, that is a cyclic subgroup of $\mathbb{Z}_{33}^*$:

$$\langle 7 \rangle = \{ \underbrace{1}_{7^0}, \underbrace{7}_{7^1}, \underbrace{16}_{7^2}, \underbrace{13}_{7^3}, \underbrace{25}_{7^4}, \underbrace{10}_{7^5}, \underbrace{4}_{7^6}, \underbrace{28}_{7^7}, \underbrace{31}_{7^8}, \underbrace{19}_{7^9}, \}$$

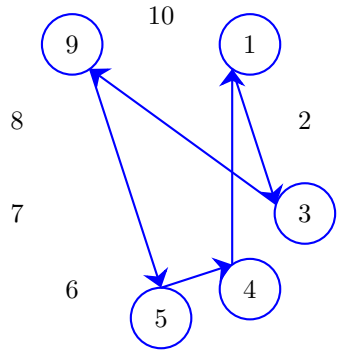
To decrypt c, we need the multiplicative inverse of e, which does not exist modulo $20 = \varphi(N)$. Indeed, no value d can lead to correct decryption since $c^d \mod N = 1^d = 1$ can only return 1 (all the powers of 1 are 1 itself) and will never return 7. So, there is no way to find the message back. Again, the problem is not with the RSA construction, but with having chosen an inappropriate exponent that has no corresponding decryption/secret key.

1.3.2 ElGamal encryption scheme

Recall that in Elgamal we work with p a large prime and g an element of $\mathbb{Z}_p^*$ which generates a cyclic (multiplicative) subgroup of prime order q.

To have an intuition, we consider the following example of parameters: let $p = 11$, $g = 3$ and $q = 5$: $g \in \mathbb{Z}_{11}^*$ where $g = 3$ is a generator of a subgroup of prime order 5, indeed

$$\begin{aligned}\langle 3\rangle = \Big\{ & 3^0 = 1 \mod 11,\\ & 3^1 = 3 \mod 11,\\ & 3^2 = 9 \mod 11,\\ & 3^3 = 3\cdot 3^2 = 3\cdot 9 = 27 = 5 \mod 11,\\ & 3^4 = 3\cdot 3^3 = 3\cdot 5 = 15 = 4 \mod 11,\\ & 3^5 = 3\cdot 3^4 = 3\cdot 4 = 12 = 1 \mod 11 \Big\}\\ = & \{1,3,9,5,4\}\end{aligned}$$



Why should we take the exponent in ElGamal in $\mathbb{Z}_q^*$?
The value q is a prime number, then $\mathbb{Z}_q^* = \mathbb{Z}_q \setminus \{0\} = \{1, 2, \ldots, q-1\}$ since every number between 0 and $q-1$ is coprime with q. The subgroup generated by g is a group of order the prime number q, $|\langle g\rangle| = q$. If we choose an exponent $b' \in \mathbb{N}$, for the Euler's theorem applyed to $\mathbb{Z}_q^*$, it holds that $g^b = g^{b'} \mod q$ where $b = b' \mod \varphi(q)$. For this reason is equivalent (and easier) to take the exponents a, b in the group $\mathbb{Z}_q^*$.

2 Security of PKE

2.1 INP-CPA Security for PKE

In this section we present **IND-CPA**: **Ind**istinguishability under **C**hosen **P**laintext **A**ttack for public key encryption schemes. We already saw IND-CPA in Module 1, and the game is essentially the same, just that now, since to encrypt we use the public key pk, $\mathcal{A}$ no longer needs access to an encryption oracle.

Definition 2. A PKE consisiting of the three PPT algorithms $(KeyGen, Enc, Dec)$ is **Indistinguishable under Chosen Plaintext Attack (IND-CPA)** if for any PPT $\mathcal{A}$

$$Pr[b = b'] \leq \frac{1}{2} + negl(n),$$

where b and b' are derived from the IND-CPA Game, defined in Figure 4. Informally, the adversary $\mathcal{A}$'s chance of winning the game is negligibly close to $\frac{1}{2}$ which is the probability of guessing the bit b at random.

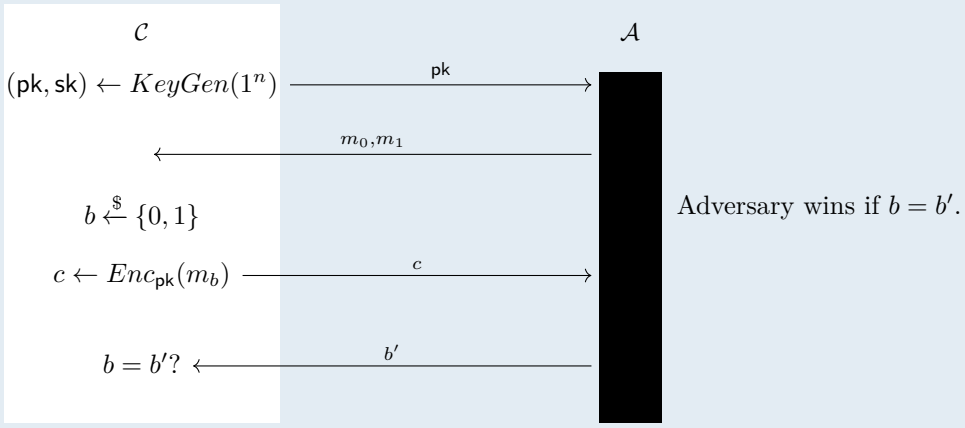


Figure 4: IND-CPA Game

The IND-CPA security game begins with the challenger who runs the key generation algorithm to obtain a pair of public key and secret key. The challenger then sends pk to the adversary. The adversary can encrypt any (polynomially many) message of its choice using pk. Once satisfied, $\mathcal{A}$ sends two messages $m_0 \neq m_1$ to $\mathcal{C}$. The challenger chooses one of the two messages at random, encrypts that message, and sends the ciphertext back to $\mathcal{A}$. The adversary $\mathcal{A}$ wins the game if it can figure out which of the two messages was encrypted with noticeably higher probability than guessing at random.

This game is very similar to IND-CPA security of symmetric encryption. The only difference is that here challenger $\mathcal{C}$ gives pk to adversary $\mathcal{A}$ instead of allowing the adversary to query the encryption oracle as in the

IND-CPA game for symmetric encryption (since in that case, the adversary does not get the symmetric encryption key, Kerckhoffs's principle).

IND-CPA and deterministic encryption. Note that a deterministic encryption scheme, such as Textbook-RSA, cannot be IND-CPA secure. Since the Adversary knows pk it can simply compute $c_0 = Enc(\mathsf{pk}, m_0)$ and $c_1 = Enc(\mathsf{pk}, m_1)$; then send m_0, m_1 to $\mathcal{C}$, and compare the challenge ciphertext c with c_0 and c_1. Since $Enc$is deterministic, $Enc(\mathsf{pk}, m_i)$ will always return the same ciphertext. So $\mathcal{A}$ can output as a guess $b' \in \{0,1\}$ such that $c_{b'} = c$ and win the IND-CPA game with probability 1 (which is noticeably larger than 1/2).

ElGamal is IND-CPA secure under DDH.

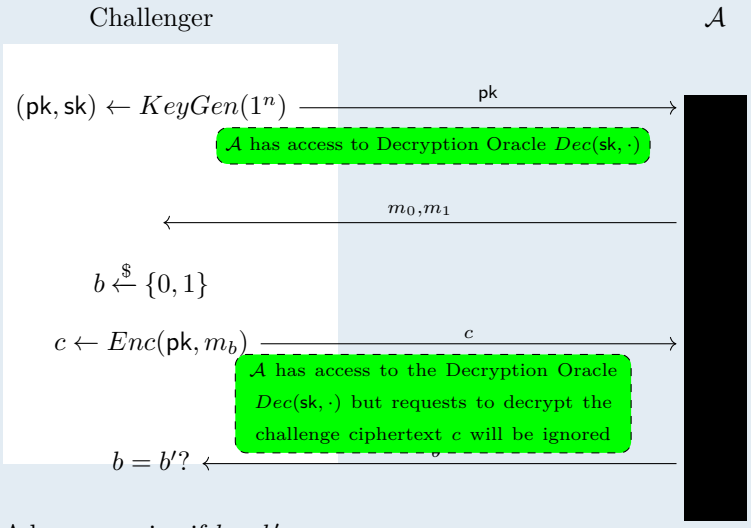
2.2 IND-CCA Security

Here, we present **IND-CCA**: **Ind**istinguishability under **C**hosen **C**ipher **A**ttack.

Definition 3. A PKE is **Indistinguishable under Chosen Cipher Attack (IND-CCA)** secure if for any PPT $\mathcal{A}$

$$Pr[b = b'] \leq \frac{1}{2} + negl(n),$$

where b and b' are derived from the IND-CCA Game defined in Figure 5. Informally, the adversary $\mathcal{A}$'s chance of winning the game is negligibly close to $\frac{1}{2}$ which is the probability of guessing the bit b at random.



Adversary wins if $b = b'$.

Figure 5: IND-CCA Game

The only difference from the IND-CPA and the IND-CCA games, is that the latter gives $\mathcal{A}$ additional access to the *Decryption Oracle* $Dec(\mathsf{sk}, \cdot)$, which means $\mathcal{A}$ can send polynomially many ciphertexts c_i requests and for each ciphertext it will receive $m_i = Dec(\mathsf{sk}, c_i)$ (this is what happens in the green rectangles in the diagram representation of the game). The adversary can use this oracle both before and after sending the two messages m_0, m_1 to $\mathcal{C}$, however, after receiving the challenge ciphertext c, the adversary is no longer allowed to query the decryption on c (otherwise $\mathcal{A}$ can trivially win the game).

IND-CCA is a stronger security notion than IND-CPA because we are giving the adversary $\mathcal{A}$ more power, i.e. the decryption oracle.

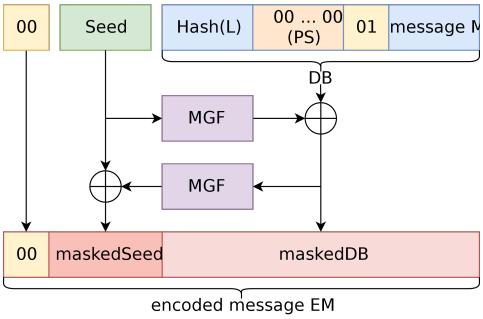
2.3 On the security of Textbook RSA and ElGamal encryption schemes

2.3.1 Textbook RSA is not IND-CPA secure

This is due to the fact that RSA is deterministic, and that it is malleable/homomorphic (more on this in section 3). One easy way to see this is via the following attack (which succeeds in winning the IND-CPA game with probability 1). The adversary chooses two distinct random messages $m_0, m_1 \in \mathbb{Z}_N^*$, encrypts each message and obtains $c_i = (m_i)^e \mod N$. Then it sends (m_0, m_1) to the challenger, gets c back. Finally $\mathcal{A}$ returns as a guess b' such that $c_{b'} = c$.

Since the IND-CCA security notion is stronger than the IND-CPA one, and RSA is not IND-CPA secure, then RSA cannot be IND-CCA secure either.

If RSA is not IND-CPA secure, why is it used everywhere? Because what is used in practice is not textbook RSA, but RSA with Optimal Asymmetric Encryption Padding (OAEP), see diagram to the right. OAEP makes RSA a randomized and non-malleability encryption scheme. The variable L in OAEP is a public label, and Seed is a random seed which provides the required randomness. MGF stands for Mask Generation Function and is a special pseudorandom function, similar to hash functions.
RSA-OAEP is proven to be IND-CCA secure under the RSA assumption (but we do not show the proof in the course).



2.4 ElGamal is not IND-CCA secure

To see this consider the following adversary algorithm for IND-CCA game with ElGamal.

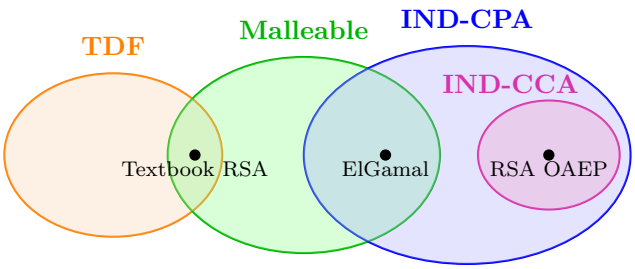
1. Set $m_0 = g$, $m_1 = g^2$.
2. Send m_0, m_1 to challenger $\mathcal{C}$.
3. Receive the ciphertext $c = (A, c')$ from $\mathcal{C}$.
4. Ask $\mathcal{C}$ to decrypt $z = (A, c' \cdot g)$. (the challenger will not ignore this request since $z \neq c$.)
5. Receive x (such that $x = Dec(\mathsf{sk}, z)$).
6. Compute $m = x \cdot g^{-1}$.
7. If $m = m_0$ return 0, otherwise ($m = m_1$) return 1.

This adversary $\mathcal{A}$ wins the IND-CCA game with probability 1.

Summary

	Textbook RSA	ElGamal	RSA-OAEP
Deterministic?	yes	no	no
Secure TDF?	yes*	no	no
Malleable?	yes	yes	no
IND-CPA?	no	yes*	yes*
IND-CCA?	no	no	yes*

* under the corresponding computational assumptions



3 Homomorphic Encryption

3.1 Malleability

We start with defining malleability,

Definition 4. A PKE scheme is **malleable** if one can modify a ciphertext and predict how does that impact the encrypted message (*knowing* pk*, but without knowledge of the message and* sk).

For example for a malleable PKE, we can for any ciphertext c that decrypts to m, produce a ciphertext c' that will decrypt to $m' = 2 \cdot m$ (without knowing m or sk). Next we show that both ElGamal and RSA are malleable.

Example 1. ElGamal is malleable, indeed given $\mathsf{pk} = B$ and c such that $Enc(\mathsf{pk}, m) = c = (A, h)$, we can compute $c' = (A, 2h \mod p)$.
Then we get $Dec(\mathsf{sk}, c') = (2 \cdot m) \mod p$ because

$$Dec(\mathsf{sk}, c') = (2h) \cdot (A^b)^{-1} = 2 \cdot (h \cdot A^{-b}) = 2m \mod p$$

Example 2. Textbook RSA is malleable, indeed given $\mathsf{pk} = (N, e)$ and c such that $Enc(\mathsf{pk}, m) = c$, we can compute $c' = 2^e \cdot c \mod N$.
Then we get $Dec(\mathsf{sk}, c') = (2 \cdot m) \mod N$ because

$$Dec(\mathsf{sk}, c') = {c'}^d = (2^e \cdot c)^d = (2^e \cdot m^e)^d = 2m \mod N$$

No malleable encryption (which allows modifying ciphertext in a predictable way without knowing sk) can be IND-CCA secure, and therefore neither textbook RSA nor ElGamal are IND-CCA secure.

Is it beneficial to have a malleable encryption scheme or not? Consider the scenario where someone presents us with two encryption schemes—one malleable and the other non-malleable—both identical in all other aspects. Which would we choose to encrypt our private data? In fact the answer is not trivial, because malleability is neither entirely a flaw nor entirely a feature. In certain cases, the ability to modify an encrypted message without decrypting is a valuable property, enabling a broader range of applications. For instance, malleable encryption can allow the computation of encrypted values, this is what we call *Homomorphic Encryption* (see Section 3)

Homomorphic encryption enables computations of ciphertexts such that the new ciphertext dectypt to the output of the same computation applied to the corresponding plaintexts. Thereby a party can compute a function of the plaintext, by applying the function to the on the ciphertexts, without having an idea of its contents or the decrypttion key. We will cover two different types of homomorphic encryption: linearly and fully homomorphic encryption. Both of these are malleable, and can therefore not be IND-CCA secure.

3.2 Linearly Homomorphic Encryption

Informally speaking, an encryption scheme is said to be **Linearly Homomorphic** (abbreviated as LHE for linearly homomomrphic encryption) if there is a known algorithm $\oplus_{\mathsf{pk}}$ such that

$$Enc(\mathsf{pk}, a) \oplus_{\mathsf{pk}} Enc(\mathsf{pk}, b) = Enc(\mathsf{pk}, (a \star b))$$

The exact meaning of the arithmetic operation $\star$ on plaintexts depends on the concrete scheme. Usually, it is related to modular arithmetic or group operations. Important to rematk is that the homomorphic operation $\oplus_{\mathsf{pk}}$ does not depend on the secret key (sk) or access to the plaintext message. This is significant because it enables data to remain confidential while still allowing certain computations to be done on the encrypted data. All LHE PKE-schems are malleable and, as a result not IND-CCA secure. However, the property of LHE can be useful in certain scenarios.

For a group $(\mathbb{G}, \star)$, the $\star$ is often called "linear operation" or "group element addition". For ElGamal, the group is $(\mathbb{Z}_p^*, \star)$ with $m_1 \star m_2 = m_1 \cdot m_2 \mod p$. and for RSA, the group is $(\mathbb{Z}_p^*, \star)$ with $m_1 \star m_2 = m_1 \cdot m_2 \mod N$. Therefore, these are called Linearly Homomorphic. Even though they correspond to modular multiplication.

In terms of multiplicative groups $(\mathbb{G}, \star)$, homomorphic multiplication means taking $h \in \mathbb{G}$ and $k \in \{0, 1, \ldots, |\mathbb{G}|\}$, and computing $h^k = \underbrace{h \star h \star \ldots h}_{k \text{ times}}$.

Example 3. Linearly Homomorphic operations on ElGamal: let $c_1 = Enc(\mathsf{pk}, m_1) = (A_1, h_1)$ encrypt m_1 and $c_2 = Enc(\mathsf{pk}, m_2) = (A_2, h_2)$ encrypt m_2 both encryptions done using the same key $K (= A^b = B^a)$. Suppose Bob is decrypting $c_1 \cdot c_2 := (A_1 \cdot A_2, h_1 \cdot h_2)$ with the private key b:

$$K = (A_1 \cdot A_2)^b = A_1^b \cdot A_2^b \qquad \mod p$$
$$m = (h_1 \cdot h_2) \cdot K^{-1} = (h_1 A_1^{-b})(h_2 A_2^{-b}) = m_1 m_2 \qquad \mod p$$

This demonstrates that the decryption of the element wise product of two ciphertexts yields product of the corresponding plaintext messages.

Example 4. Linearly Homomorphic operations on Textbook RSA: let $Enc(\mathsf{pk}, m_1) = c_1$ and $Enc(\mathsf{pk}, m_2) = c_2$ be RSA ciphertexts, encrypten for Alice whose private key is (N, d). Let's see what happens if Alice decrypts $c' = c_1 \cdot c_2 (= m_1^e m_2^e)$:

$$Dec(\mathsf{sk}, c') = (c')^d = (c_1 c_2)^d = (m_1^e \cdot m_2^e)^d = (m_1 \cdot m_2)^{ed} = m_1 m_2 \qquad \mod N$$

3.3 Fully Homomorphic Encryption

While linearly homomorphic encryption supports only one type of operation on cipher/plain-texts (typically addition), Fully Homomorphic Encryption (FHE) is designed to allow both additions and multiplications.

In FHE, the entire Boolean algebra is implemented within the encryption scheme, making it possible to perform any operation that can be represented by Boolean circuits.

FHE schemes are significantly more versatile than LHE schemes but are also more complex and demand substantially higher computational resources. Like LHE, FHE schemes are inherently malleable and, consequently, cannot achieve IND-CCA security. There are relaxations of this security notions that are achieved by some homomorphic encryption constructions, but we will not discuss them in this course. FHE can be used to outsource computations on confidential data to an untrusted party. Two such examples are:

- A Cloud that can process people's private data without being able to peek at the data.
- Multi-Party Computation where parties jointly perform operations on their data without any of them being able to see in plaintext the data they are operating on.

Fully Homomorphic schemes allow the following two operations:

Addition : Given $Enc(\mathsf{pk}, m_1)$ and $Enc(\mathsf{pk}, m_2)$ there is a procedure to compute $Enc(\mathsf{pk}, (m_1 + m_2))$.

Multiplication : Given $Enc(\mathsf{pk}, m_1)$ and $Enc(\mathsf{pk}, m_2)$, there is a procedure to compute $Enc(\mathsf{pk}, (m_1 \cdot m_2))$.

4 How to compute the inverse in $\mathbb{Z}_N^*$ efficiently

The Euler's totient function has a surprising application for the groups of residues modulo. As shown in the Euler's theorem below, raising any $x \in \mathbb{Z}_M^*$ to the power of $\varphi(M)$ is guaranteed to produce 1.

Theorem 1 (Euler's Theorem)**.** Let $M > 0$ and $x \in \mathbb{Z}_M^*$, then $x^{\varphi(M)} = 1 \mod M$.

And the following Fermat's Little Theorem follows from Euler's theorem when $M = p$ is prime.

Theorem 2 (Fermat's Little Theorem)**.** Let p be a prime and $x \in \mathbb{Z}_p^*$, then $x^{p-1} = 1 \mod p$.

Compute inverse using Euler's Theorem The important observation here is that if you raise x to the power $\varphi(M) - 1$, you get something that is guaranteed to collapse to 1 after being multiplied by x once more.

In other words, if $y = x^{\varphi(M)-1}$, then $y \cdot x = x^{\varphi(M)} = 1 \mod M$. So y is the inverse of x modulo M, $y = x^{-1} \mod M$.

For a more concrete example, consider the cases below with p and q being primes.

$\underline{M = p}$**:** For any $x \in \mathbb{Z}_p^*$, x^{p-2} is the multiplicative inverse of x, indeed $x^{(p-1)-1} = x^{p-2} = x^{-1} \mod p$.

$\underline{M = p \cdot q}$**:** For and any $x \in \mathbb{Z}_M^*$, x^{pq-p-q} is the multiplicative inverse of x, indeed $x^{(p-1)(q-1)-1} = x^{pq-p-q} = x^{-1} \mod M$.

Euler's totient function gives us a convenient method to invert values in $\mathbb{Z}_M^*$ using the formulas above. This method crucially relies on two things: knowing $\varphi(M)$; being able to efficiently raise x to the power $\varphi(M)$. We achieve first by knowing the factorization of M (if we do not know it, the method does not apply).

How does one efficiently raise x to some power i modulo M? An obvious naïve approach one could try is i times iteratively multiplying x by itself modulo M. This will require i modular multiplications. When $i = \varphi(M)$ and M is very large (say $M > 2^{2048}$), this quickly becomes infeasible. Luckily for us, there exists a faster method for computing $x^i \mod M$ (the mod operator here denotes modulus operation like in C++/Java/Python programming languages) called *Binary Exponentiation*. It does so using at most $2\lceil \log_2 M \rceil$ modular multiplications. In practice, Binary Exponentiation works in this way

$$\text{binExp}(x, i, M) = \begin{cases} 1 & \text{if } i = 0, \\ \text{binExp}(x, i/2, M)^2 \mod M & \text{else if i is even,} \\ \text{binExp}(x, i-1, M) \cdot x \mod M & \text{else if i is odd.} \end{cases}$$

Binary Exponentiation reduces the total number of multiplications using the property that $x^{2k} = (2^k)^2$. So if you compute 2^k using a recursive step, you can go 2^{2k} using only one multiplication.

To reiterate, here's how you invert values modulo M for some of the very common M values:

$\underline{M = p}$**:** For any $x \in \mathbb{Z}_p^*$, $x^{-1} = \text{binExp}(x, p-2, p)$.

$\underline{M = p \cdot q}$**:** For and any $x \in \mathbb{Z}_M^*$, $x^{-1} = \text{binExp}(x, (p-1)(q-1)-1, pq)$.

4.1 Finding inverses when the factorization is unknown

The method we have shown in the previous section is efficient, but it heavily relies on the knowledge of the factorization of M. This always works for ElGamal encryption scheme, since we need to compute the inverse of an element in $\mathbb{Z}_p^*$. The problem appears with RSA in which we need to compute the inverse of an element in the group $\mathbb{Z}_{\varphi(N)}^*$. In this case, to apply the method shown in the previous section we need to either know the factorization of $\varphi(N)$ or compute the value $\varphi(\varphi(N))$, but both these are hard problems. In this section, we will see another method that does not rely on any special properties of M.

Theorem 3 (Bézout's Identity)**.** Let x and y be positive integers, and $d = \gcd(x, y)$. Then, there exist (possibly negative) integers s and t such that

$$xs + yt = d.$$

Bézout's Identity says that for any x and y, there exist such coefficients s and t that adding them with these coefficients will give $\gcd(x, y)$. It may not be obvious right away why this is useful at this point. This should become clearer as we gradually build up the method below.

EEA(x, y)

if $x = 0$ **then**
  return $(y, 0, 1)$
$(d, s', t') := \text{EEA}(y \mod x, x)$
$s := t' - \lfloor y/x \rfloor \cdot s'$
$t := s'$
return (d, s, t)

The Extended Euclidean Algorithm (EEA) computes Bézout coefficients s and t for any positive integers x and y. It is very efficient, and can work for large M values ($M > 2^{2048}$) with sub-second running time of regular desktop computers.

How can we exploit Bézout's Identity and EEA to find the inverse of $x \in \mathbb{Z}_M^*$ modulo M? Suppose we applied the identity to x and $y = M$. The identity says that there exist s and t such that

$$xs + Mt = \gcd(x, M).$$

We know that x and M are co-prime, so let's replace $\gcd(x, M)$ with 1.

$$xs + Mt = 1.$$

Now reduce both sides of the identity modulo M.

$$xs + \underbrace{Mt}_{0 \mod M} = 1 \mod M$$
$$xs = 1 \mod M.$$

That means s is the inverse of x modulo M!

We now summarise the above and state the complete method of computing $x^{-1} \mod M$ for any $x \in \mathbb{Z}_M^*$ is given by these two steps:

1. Compute $(d, s, t) = \text{EEA}(x, M)$. (The d here will always equal 1, since $x \in \mathbb{Z}_M^*$ implies x, M are co-prime)
2. return $s \mod M$.

TDA352/DIT250: Cryptography LP2, Fall 2023

Lecture Notes on

The Signal Protocol

Lecturer: Elena Pagnin Lecture #8 on Nov 24

Scribes: Elias Ekroth & Adrian Perez Keilty Last update November 24, 2023

Contents

1 Context and Relevance

Instant messaging (IM) services such as WhatsApp, Facebook messenger or Telegram host overall several billion users worldwide, and IM is becoming increasingly embedded into our daily communication. From a cryptographic point of view there are several features that are desirable when using IM:

1. Confidentiality: No one else besides the communicating parties should be able to read the conversation.
2. Message integrity: No one else should be able to modify the conversation.
3. Authentication: Sender and receiver should be guaranteed that they are communicating with each other.
4. Forward secrecy: If at a time t^* the conversation has been compromised, messages prior to t^* should remain secret.
5. Backward secrecy (or post-compromise security): If at a time t^* the conversation has been compromised, there exists a time $t' > t^*$ such that messages sent after t' regain secrecy.

The last feature is usually achieved as a result of a key healing process, i.e, new fresh keys have been securely exchanged to encrypt future messages (see Figure 1).

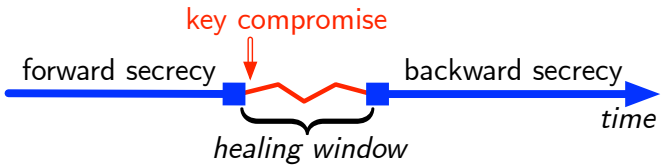


Figure 1: Key healing.

Among all IM protocols, Signal distinguished itself by achieving all the security goals above. In these lecture notes, we will go through the inner workings of the Signal protocol and show how these security goals are met.

2 Signal's Building Blocks

Signal works by cleverly integrating special cryptographic tools within a secure ping-pong-like mechanism between two communicating parties, Alice and Bob. To understand the protocol we first need to learn about key derivation functions (KDFs), authenticated encryption with associated data (AEAD) and the extended triple Diffie Hellman key agreement protocol (X3DH).

2.1 Diffie Hellman Key Exchange (DH)

Suppose Alice and Bob want to securely agree on a shared secret key over an insecure channel. In the Diffie Hellman (DH) key exchange, Alice and Bob first agree on a cyclic group $\mathbb{G}$ of prime order q, and a generator $g \in \mathbb{G}$ which are publicly known, and then proceed as follows:

1. Alice picks a secret random natural number $a \in \mathbb{Z}_q^*$, and sends the element $A = g^a$ to Bob.
2. Analogously, Bob picks a random $b \in \mathbb{Z}_q^*$, and sends $B = g^b$ to Alice.
3. Alice computes $B^a = (g^b)^a = g^{ba}$ while Bob computes $A^b = (g^a)^b = g^{ab}$

Due to the commutativity in exponentation, both parites end up with the same (secret) value $K = g^{ab} \in G$. As long as there is no efficient algorithm for determining g^{ab} given g, g^a, and g^b (Diffie-Hellman problem), Alice and Bob have successfully agreed on a key only known by them.

Notation: To denote a secret K obtained during a DH key exchange, i.e, $K = A^b = B^a$, we write $K = \mathsf{DH}(A, B)$. Notice that even if $\mathsf{DH}(\cdot, \cdot)$ takes in public parameters, it is not a publicly computable function!

2.2 Key Derivation Function (KDF)

A key derivation function (KDF) is a deterministic algorithm that takes as input a previously generated key and one or more other inputs and outputs one or more fresh secret keys to use at a later stage. A KDF has the following properties:

- *Resilience*: The output keys appear random to an adversary $\mathcal{A}$ without knowledge of the KDF keys. This is true even if the adversary can control the KDF inputs.
- *Forward security*: Output keys from the past appear random to $\mathcal{A}$ who learns the KDF key at some point in time.
- *Break-in recovery*: Future output keys appear random to $\mathcal{A}$ who learns the KDF key at some point in time, provided that future inputs have added sufficient entropy.

Intuitively, a KDF preserves the entropy of the input keys by generating new fresh keys in a one-way fashion (see for example HKDF).

2.3 Authenticated Encryption with Associated Data (AEAD)

AEAD is an authenticated encryption scheme (AE) which allows additional non-encrypted data to be included in the message, that can be also authenticated. Specifically, given a plaintext m and associated data AD, AEAD outputs a ciphertext c, an unmodified copy of AD and a authentication tag t (or MAC) to authenticate the first two:

$$\mathsf{AEAD.Encrypt}_{\mathsf{k}}\left(m, \mathsf{AD}\right) = (c, \mathsf{AD}, t)$$

In the Signal protocol, this feature will be useful to a receiver for reconstructing key material from the associated data for a later use.

2.4 Extended Triple Diffie Hellman Key Agreement Protocol (X3DH)

In the IM setting, two communication parties should be able to send messages and authenticate eachother even if one of them is offline. X3DH is used during the session (chat) initialization phase of the Signal protocol for this purpose and to establish a shared secret key. This key can then be used during the lifetime of that session. The following is a compressed version of X3DH that has been described in [1].

The X3DH protocol involves three parties: Alice, Bob, and a server.

- Alice wants to send Bob some initial data using encryption, and also establish a shared secret key ms_{AB} which may be used for further secure communication (ms_{AB} is often called "master secret" even though it is rather used as a seed key to kick-start the communication, and should be deleted after use).
- Bob wants to allow parties like Alice to establish a shared key with him and send encrypted data. However, Bob might be offline when Alice attempts to do this. To enable this, Bob has a relationship with some server.
- The server can store messages from Alice to Bob which Bob can later retrieve. The server also lets Bob publish some data which the server will provide to parties like Alice.

X3DH consists of three phases:

1. Server key upload:

   Bob uploads a set of elliptic curve[1] public keys to the server, containing:

   - An identity public key idpk_B
   - A mid-term signed public key mtpk_B
   - A set of one-time public keys $\{\mathsf{otpk}_B^{(i)}\}_{i=1}^{N}$

   Bob only needs to upload his identity key to the server once. However, Bob may upload new one-time keys at other times (e.g. when the server informs Bob that the number of one-time keys is running low) and a new mid-term signed public key at some interval (e.g. once a month) which will replace the ones that have already been used.

2. Sending the initial message:

   To perform an X3DH key agreement with Bob, Alice contacts the server and fetches idpk_B, mtpk_B and one of the one-time keys otpk_B.

[1] Either X25519 or X448

After providing Bob's one-time key, the server deletes it. Alice verifies the signed mid-term key for authentication and aborts the protocol if verification fails. Otherwise, Alice proceeds by generating an ephemeral key pair with public key epk_A. She then computes the (future) shared secret

$$\mathsf{ms}_{AB} = \mathsf{DH}(\mathsf{idpk}_A, \mathsf{mtpk}_B)) \,||\, \mathsf{DH}(\mathsf{epk}_A, \mathsf{idpk}_B) \,||\, \mathsf{DH}(\mathsf{epk}_A, \mathsf{mtpk}_B) \,||\, \mathsf{DH}(\mathsf{epk}_A, \mathsf{otpk}_B)$$

which is later formatted into

$$\mathsf{ms} = \mathsf{KDF}(\mathsf{ms}_{AB})$$

for some KDF, where DH stands for the Diffie-Hellman key exchange on an elliptic curve. Immediately after this she deletes her ephemeral private key as well as the DH outputs.

Alice then sends her identity and ephemeral keys idpk_A, epk_A, the index of Bob's one-time key that she used, and an AEAD using ms as an encryption key, of an initial message containing the additional data $\mathsf{AD} = \mathsf{Encode}(\mathsf{idpk}_A) \,||\, \mathsf{Encode}(\mathsf{idpk}_B)$.

After this, Alice may continue using ms or keys derived from ms within the post-X3DH protocol for future communication with Bob.

3. Receiving the initial message:

   Upon receiving Alice's initial message, Bob loads his private keys idsk_B, mtsk_B and otsk_B (corresponding to whichever mid-term signed key and one-time key Alice used) and together with idpk_A and epk_A, he reconstructs ms and AD. Finally, Bob attempts to decrypt the initial ciphertext using ms and AD. If the initial ciphertext fails to decrypt, then Bob aborts the protocol and deletes ms. Otherwise, the protocol is completed. Bob may then continue using ms or keys derived from ms within the post-X3DH protocol for further communication with Alice.

3 The Double Ratchet Mechanism

A "ratchet" in the cryptographic sense is an algorithm that only moves forward, one step at a time and with no repetitions (as opposed to a mechanical ratchet). Following the Signal protocol, during their IM communication Alice and Bob will make use of two ratchets. In order to define these properly, we first map each message to a two-dimensional stage (x, y) where x is incremented each time Alice or Bob start a new sequence of messages and y keeps track of the number of messages sent during each level x as shown in Figure 2.

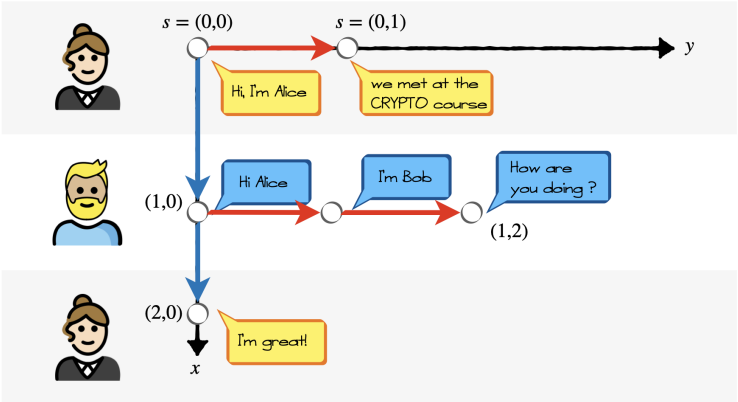


Figure 2: Asynchronous message mapping.

3.1 The Asymmetric Ratchet

An asymmetric ratchet, just like asymmetric encryption, makes use of the public keys of the communicating parties. In the Signal protocol, at each level x of the conversation a new assymetric ratchet consisting of a KDF will be triggered. This KDF, denoted KDF_1 [1] will take as input the root key of the previous level $\mathsf{rk}^{(x-1)}$ and a freshly exchanged Diffie-Hellman key $\mathsf{DH.key}^{(x)}$ and will output the root key for the current level $\mathsf{rk}^{(x)}$ together with a chain key $\mathsf{ck}^{(x,0)}$. The latter will serve as a seed to encrypt the messages located at stages $(x, 0), (x, 1), \ldots$ as shown in Figure 3.

[1] In practice, $\mathsf{KDF}_1(\cdot,\cdot) = \mathsf{HKDF}(\cdot,\cdot,\mathsf{const}_2)$.

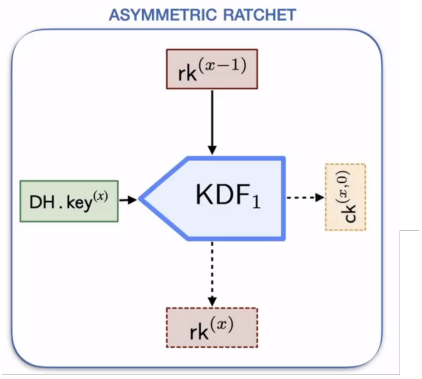


Figure 3: Asymmetric ratchet.

3.2 The Symmetric Ratchet

The symmetric ratchet kicks in every time the current sender sends a new message (while the responder is silent). It takes as input a chain key $\mathsf{ck}^{(x,0)}$ and runs a KDF on it, the KDF is denoted as KDF_2 [2]. It will output a message key $\mathsf{k}^{(x,0)}$, to encrypt the message at $(x, 0)$, and a chain key $\mathsf{ck}^{(x,1)}$, which in turn will be used as input for the next symmetric ratchet (if the sender keeps producing messages before receiving a reply). The symmetric ratchet will execute for each message along the y axis until the receiver replies, which will set $y \leftarrow 0$ and $x \leftarrow x + 1$, triggering both ratchets once again (see Figure 3).

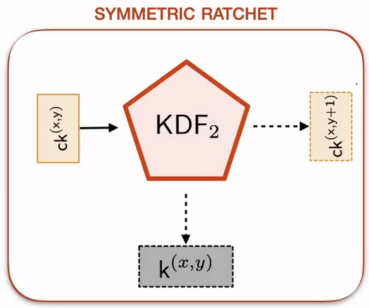


Figure 4: Symmetric ratchet.

As described, both ratchets are linked via an initial chain key $\mathsf{ck}^{(x,0)}$. This combination is commonly called the 'Double Ratchet Mechanism'. During the Signal protocol Alice and Bob will take turns in initializing an iteration of this process for each level x.

4 The Signal Protocol

The Signal protocol is used by the Signal app, WhatsApp, Skype, Facebook Messenger, and many other IM apps.

In what follows, we will present the main phases of the protocol (see [2] for a full specification). We divide the description in three steps:

- A one-time user registration (run once in the lifetime of a user in the system).
- A new-session initialization (run once per new session initiated by the user).
- The double ratchet mechanism for messaging (run every time one of the users replies to received message).

4.1 Keys

In Signal, each user holds a set of keys: one long-term key (referred to as identity key), a mid-term key (which is periodically updated) and several one-time use keys that will be used only to initiate new sessions (chats). Keys employed only to set up new sessions are highlighted with the symbol ★ (see [3])

[2] in practice, $\mathsf{KDF}_2(\cdot) = (\mathsf{HMAC}(\cdot, 0), \mathsf{HKDF}(\mathsf{HMAC}(\cdot, 1), \mathsf{const}_1))$

Name	Key(s)	Type	Usage
Identity key-pair★	$\{\mathsf{idpk}_U, \mathsf{idsk}_U\}$	Asymmetric	Long-term
Mid-term key-pair★	$\{\mathsf{mtpk}_U, \mathsf{mtsk}_U\}$	Asymmetric	Mid-term
One-time key-pair★	$\{\mathsf{otpk}_U, \mathsf{otsk}_U\}$	Asymmetric	One-time
Ephemeral key-pair★	$\{\mathsf{epk}_U, \mathsf{esk}_U\}$	Asymmetric	One-time
Ratchet key-pair	$\{\mathsf{rchpk}_U, \mathsf{rchsk}_U\}$	Asymmetric	One-time
Master secret	ms	Symmetric	One-time
Root key	$\mathsf{rk}^{(x)}$	Symmetric	One-time
Chain key	$\mathsf{ck}^{(x,y)}$	Symmetric	One-time
Message key	$\mathsf{k}^{(x,y)}$	Symmetric	One-time

Table 1: List of keys used in Signal.

All asymmetric key pairs are of the form $sk \xleftarrow{\$} \mathbb{Z}_p$ (where p is a large prime number) and $pk = g^{sk} \in \mathbb{G}$, where g is a generator of the group $\mathbb{G}$, $\mathbb{G}$ has prime order p, and the discrete logarithm and the Computational Diffie-Hellman (CDH) assumptions hold in $\mathbb{G}$.

4.2 User Registration

At installation, each user registers to the Signal server by providing a unique identifier (their phone number). The user also generates several key pairs and uploads to the Signal server their long-term identity public key idpk_U, their mid-term public key mtpk_U, and a number N of one-time keys $\{\mathsf{otpk}_U^{(i)}\}_{i=1}^{N}$.

4.3 Session Initialization (X3DH handshake)

Suppose Alice wants to initiates a new session with Bob. In this case Alice sends a request to the Signal server by providing Bob's unique identifier (e.g, Bob's's phone number). The server then fetches the key material uploaded by Bob during the user registration phase, and returns to Alice his identity, mid term, and one-time public keys $\mathsf{idpk}_B, \mathsf{mtpk}_B, \mathsf{otpk}_B$. To prevent using the same one-time key to initiate other sessions, the Signal server will delete the otpk_B it sent to Alice from Bob's public key material. Now Alice can use her identity *secret* key idsk_A (the Signal server holds the corresponding public key idpk_A) and a freshly generated ephemeral key pair $(\mathsf{esk}_A, \mathsf{epk}_A)$ to compute the (future) shared master secret key ms_{AB} between her and Bob. Concretely, ms_{AB} is obtained by concatenating four DH keys, as follows:

$$\mathsf{ms}_{AB} = {\mathsf{mtpk}_B}^{\mathsf{idsk}_A} \| {\mathsf{idpk}_B}^{\mathsf{esk}_A} \| {\mathsf{mtpk}_B}^{\mathsf{esk}_A} \| {\mathsf{otpk}_B}^{\mathsf{esk}_A}$$

Following X3DH, the shared master secret will be formatted into

$$\mathsf{ms} = \mathsf{KDF}(\mathsf{ms}_{AB})$$

for a public KDF, and used in this form to initiate the double ratchet mechanism.

Until now Bob is unaware of the fact that Alice has been initializing a session. This will be evident only after Alice sends the first message (explained in the next step) *and* Bob gets online, i.e., Bob connects to the Signal server and fetches incoming messages. Clearly, as long as Bob obtains Alice's identity public key and the ephemeral public key epk_A, it is possible for him to reconstruct the same master secret ms_{AB} as:

$$\mathsf{ms}_{AB} = {\mathsf{idpk}_A}^{\mathsf{mtsk}_B} \| {\mathsf{epk}_A}^{\mathsf{idsk}_B} \| {\mathsf{epk}_A}^{\mathsf{mtsk}_B} \| {\mathsf{epk}_A}^{\mathsf{otsk}_B}.$$

The equality is immediate once it is noticed that each part of ms_{AB} is a DH key, i.e.,

$$pk_B^{sk_A} = (g^{sk_B})^{sk_A} = (g^{sk_A})^{sk_B} = pk_A^{sk_B}\ .$$

4.4 Ratchet Initialization and Messaging

Immediately after initializing the session, the first asymmetric ratchet will be executed with the formatted master secret ms (for now only known by Alice) and a fresh DH-based ratchet key pair $(\mathsf{rchsk}_A, \mathsf{rchsk}_A)$ created by Alice and used on Bob's mid term public key mtpk_B as a source of randomness. This will output the first root key value $\mathsf{rk}^{(0)}$ and chain key $\mathsf{ck}^{(0,0)}$. This chain key will be used as the input for the symmetric ratchet to produce the first message key $\mathsf{k}^{(0,0)}$ as shown in Figure 5.

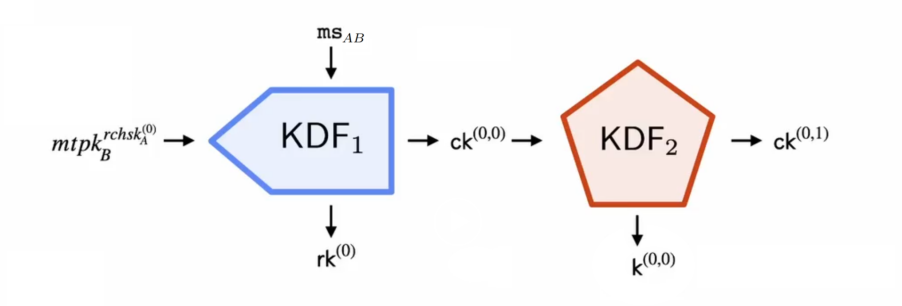


Figure 5: Ratchet initialization and first message key

$\mathsf{k}^{(0,0)}$ will in turn be used to encrypt the first message $m^{(0,0)}$ (*"Hi I'm Alice"* in Figure 2) via AEAD:

$$\mathsf{AEAD}_{\mathsf{k}^{(0,0)}}\left[m^{(0,0)}, \mathsf{AD} = \{\mathsf{idpk}_A, \mathsf{idpk}_B, \mathsf{epk}_A, \mathsf{rchpk}_A^{(0)}\}\right] = c^{(0,0)}$$

for some symmetric encryption algorithm E. Bob will use the additional data (AD) to reconstruct

- ms from idpk_A and epk_A,
- the randomness used on his mid term public key mtpk_B from $\mathsf{rchpk}_A^{(0)}$,
- the message key $\mathsf{k}^{(0,0)}$ as a result of a correct ratchet initialization from the two previous reconstructions.

After Bob has finished decrypting the rest of the messages of level $x = 0$ and decides to reply, he will in turn start the next iteration of double-ratchets by following the same steps Alice did before, and this will continue in a ping-pong fashion. After and Bob have initialized a session for IM, the assymetric and symmetric ratcheting (or double ratchet mechanism) will look as depicted in Figure 6.

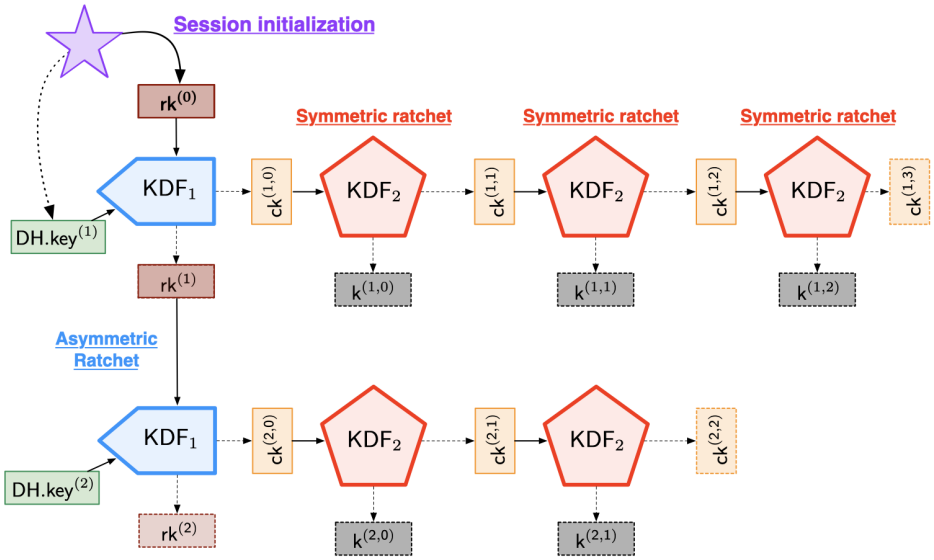


Figure 6: Double ratchet mechanism iterations.

5 Security Guarantees and Attacks

By exchanging ciphertexts via the Signal server and decrypting the plaintexts on their devices via AEAD, Alice and Bob's messages achieve confidentiality and data integrity (messages can only be corrupted by forging the AEAD's underlying MAC). Moreover, party authentication is guaranteed from the X3DH key agreement protocol, which also enables asynchronicity (Bob may be offline when Alice sends him messages).

In regards to forward and backward secrecy, we will analyse attack scenarios where the adversary $\mathcal{A}$ has access to the device of one of the communicating parties and can even cut off one of them from an internet connection.

5.1 Reveal Attack

In this type of attack we consider the scenario in which $\mathcal{A}$ tampers with the device of one of the communicating parties, e.g, Bob, thereby accessing the key material used at a certain stage of the conversation (x, y).

In particular, $\mathcal{A}$ extracts $\mathsf{k}^{(x,y)}$, $\mathsf{ck}^{(x,y)}$ used in the symmetric ratchet, and rk^x and rchsk^x which correspond to the input of the current assymetric ratchet [2] used by Bob at the start of level x. This information can then be exploited in the following ways:

- $\mathcal{A}$ computes the next message and chain keys alongside the symmetric ratchet and therefore decrypting all subsequent messages $(x, y + i)$ until Alice starts a new level $(x + 1, 0)$ (recall that KDF_2 only takes as input the current chain key $\mathsf{ck}^{(x,y)}$ so this process is purely deterministic).
- $\mathsf{DH.key}^{(x+1)}$ can be effectively reconstructed by $\mathcal{A}$ given Bob's private ratchet key from the previous level rchsk^x and the root key rk^x. $\mathsf{DH.key}^{(x+1)}$ and rk^x can then be fed to KDF_1 to produce the first chain key $\mathsf{ck}^{(x+1,0)}$ and again decrypt all follow up messages sent by Alice.

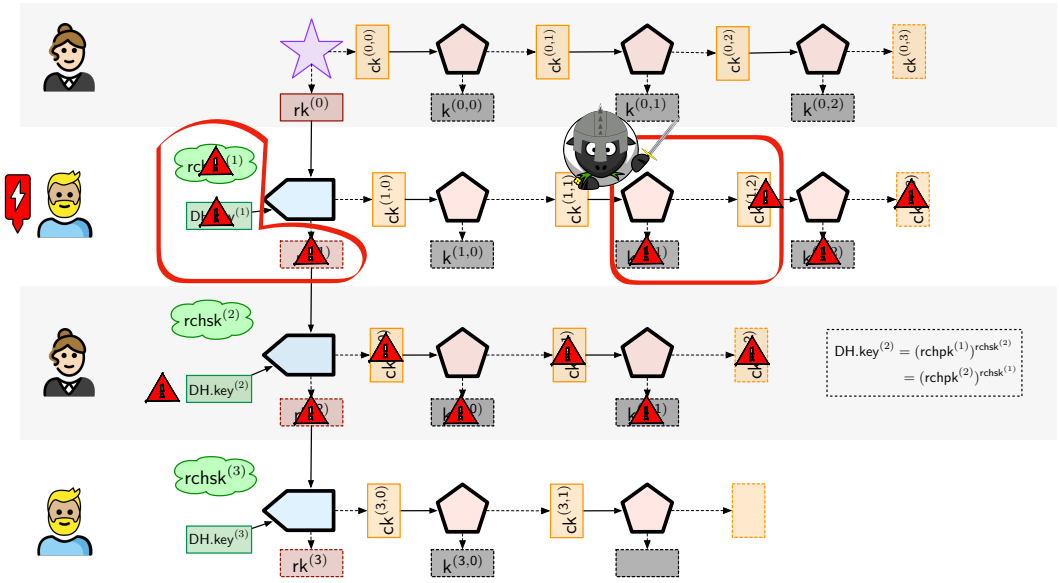


Figure 7: Healing window in a reveal attack (passive adversary)

In other words, if at a given point the set of keys of one of the communicating parties is compromised during the protocol, then two levels of messages (two asymmetric ratchets) are revealed (see Figure 7).

5.2 Reveal & Hijack Attack

The reveal attack can be used as a kick-off for a more powerful attack. After obtaining the set of keys of Bob's device, $\mathcal{A}$ could reroute the communication to her own device, exploiting the fact that there's no authentication step needed in order to keep communicating. Indeed, after the session initialization, the protocol only requires the sender to possess the previous chain key, root key and message key to validate follow up messages. It turns out that $\mathcal{A}$ can choose to hijack any of the two sessions:

1. Hijacking Alice's session: After performing a reveal attack on Bob's device at a stage (x, y), $\mathcal{A}$ can block Alice from the communication and impersonate her by sending a *different* ratchet private key $\widetilde{\mathsf{rchsk}}^{(x+1)}$ during the next DH key exchange. This can be done since $\widetilde{\mathsf{rchsk}}^{(x+1)}$ is encrypted via AEAD with the correct message key that $\mathcal{A}$ has accessed during the reveal attack. After Bob has accepted $\mathcal{A}$'s key instead of Alice's, Alice will end up with a different DH key than Bob at level $x+1$ and consequently will be left out of the communication (all subsequent generated ratchet keys will be out of sync with Bob's ones) while Bob will believe he's still communicating with Alice.

[2] note that firstly, $\mathsf{k}^{(x,y)}$ and $\mathsf{ck}^{(x,y)}$ need to be stored on Bob's device in order for him to encrypt or decrypt messages and secondly, rk^x and rchsk^x also need to be stored for computing the output of the next assymetric ratchet iteration ($\mathsf{DH.key}^{(x+1)}$ is derived from rchsk^x and rchpk^{x+1})

2. Hijacking Bob's session: In this case, instead of blocking Alice at level $x+1$, $\mathcal{A}$ waits until Alice has triggered the next assymetric ratchet to then block Bob and start level $x+2$ of the communication by performing the same attack described before (recall that the healing window for the reveal attack is 2 assymetric ratchets) this time leaving Bob out of sync without Alice realizing it.

While this attack is possible, it is still hard to achieve since $\mathcal{A}$ would need either physical access to Bob's device or malware capable of accessing it remotely and finding the keys. Moreover, even if $\mathcal{A}$ is successful finding the set of keys, 2 assymetric ratchets can take place before $\mathcal{A}$ manages to block any of the two sessions.

References

[1] Moxie Marlinspike. The X3DH Key Agreement Protocol, 2016.

[2] Moxie Marlinspike. The Double Ratchet Algorithm, November 2016.

[3] Boel Nelson, Elena Pagnin, and Aslan Askarov. To Signal or Not to Signal? Layering Traffic Analysis Resistance on Secure Instant Messaging, May 2023.

TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Secret Sharing and Commitments

Lecturer: Elena Pagnin Lecture 9 on Dec 3

Scribes: Willem Brahmstaedt, Hanna Ek & Lucia Lavagnino Last update December 20, 2024

Contents

1 Secret Sharing Scheme

Secret Sharing is a way to split a secret into multiple shares such that each individual shares hides the secret perfectly. For an intuitive understanding of the concept, one could view the key as a treasure map that is split into multiple pieces so that if one piece of the map is compromised the security of the treasure is not compromised.

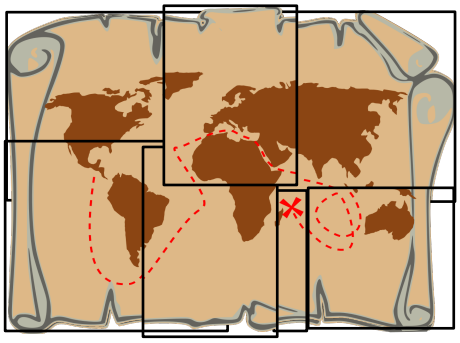


Figure 1: Key seen as a treasure map

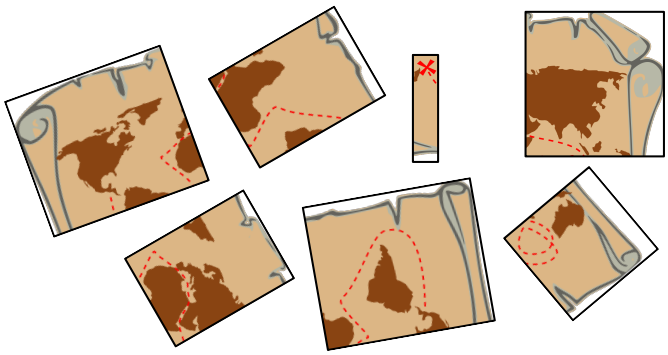


Figure 2: Splitting the key into multiple pieces

What are the implications of "splitting" a cryptographic key in this way? We want to ensure that all pieces are required to reconstruct the key. We aim to *split* the key so that its reconstruction is only possible when all the pieces are combined.

Vulnerabilities of key storage In Figure 3, we can see a few examples of vulnerabilities that illustrate this need for secret sharing Occasionally, as shown in Figure 3(a), vulnerabilities in cryptographic libraries have posed significant threats: for instance, a few years ago, dozens of such libraries were found to be insecure. Using a vulnerable cryptographic library may result in inadvertently leaking our secret key. If the secret key is compromised, the cryptographic attack essentially originates from within, undermining the very security we rely upon. All the security mechanisms we observe remain effective only if the secret key is hidden from adversaries.
Another example, depicted in Figure 3(b), involves cryptographic wallets. In these cases, attackers have managed to steal secret keys, enabling unauthorized access.

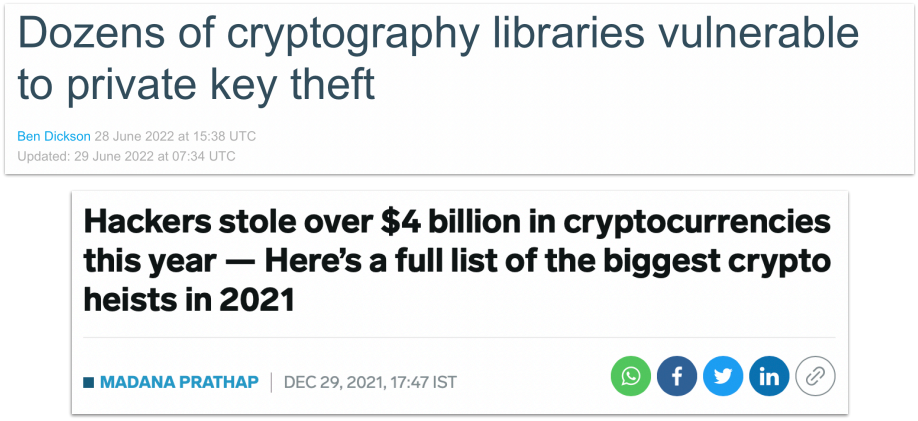


Figure 3: (a) and (b)

These examples highlight the risk of storing the key in a single location. Doing so exposes us to the types of attacks described above, thus secret sharing gives us a crucial mechanism for enhancing security. The best way to store a secret key involves ensuring both accessibility (by honest users) and security (non-accessibility by unauthorized parties). That is, on one hand we would like to store copies of the key in different locations, so that if one location is no longer accessible/available (e.g., a USB drive stops working, or a phone is out of battery) it is still possible to retrieve and use the key from another location. On the other hand, from a security perspective, we want to avoid having too many copies of the key, as each additional copy increases the risk of it being compromised and falling into the hands of an adversary $\mathcal{A}$. It seems that accessibility and security create contradictory demands, but as we will see, there is a way to fulfill both properties simultaneously using a cryptographic tool called (threshold) secret sharing.

The intuition of a secret sharing scheme is described in figure 4, it involves a dealer, who possesses the secret and distributes shares of it to various parties. These parties are commonly known as shareholders, because they each hold a portion of the distributed shares. The secret can only be reconstructed if the shareholders collaborate. To achieve this, the secret cannot simply be divided into separate pieces, as some pieces might inadvertently reveal more information about the secret than others.

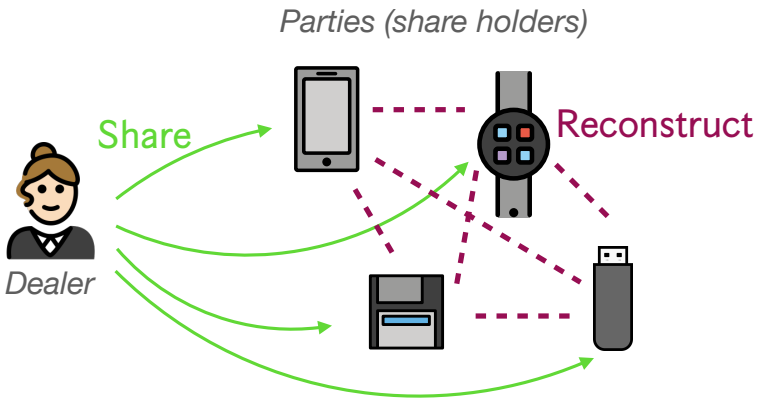


Figure 4

We can define a secret sharing scheme through two efficient algorithms:

- $\mathsf{Share}(s) \to \{s_1, \ldots, s_n\}$: Given in input a secret s, this randomized algorithm produces n shares of the secret.
- $\mathsf{Reconstruct}(s_1, s_2, \ldots, s_n) = s$: Given in input all n shares, this deterministic algorithm, computes the original secret s.

Clearly, the Reconstruct algorithm must be deterministic. Otherwise, every time it is run on the same set of shares, it might produce a different secret, whereas the goal is to reconstruct the same s consistently.

1.1 Threshold Secret Sharing

The question arises: can this approach be improved? Specifically, is it possible to design a mechanism such that any subset of t shares can be used to reconstruct the secret s? This would allow greater flexibility and fault tolerance in secret sharing schemes.

Definition 1. Threshold Secret Sharing: A t-out-of-n secret sharing scheme consists of two efficient algorithms:

- $\mathsf{Share}(s) \to \{s_1, \ldots, s_n\}$ is possibly randomized, it takes as input a secret s, and outputs n shares of the secret.
- $\mathsf{Reconstruct}(s_{i_1}, s_{i_2}, \ldots, s_{i_t}) = s$ is deterministic, it takes as input a set of $t \leq n$ distinct shares (including their indexes), and computes the secret s.

Formally a secret sharing scheme has the following properties.

1. $t-$**correctness**: any set of t parties can reconstruct s, i.e., for all $\{i_1, \ldots, i_t\} \subset \{1, \ldots, n\}$

$$Pr[\mathsf{Reconstruct}(s_{i_1}, s_{i_2}, ..., s_{i_t}) = s \mid (s_1, ..., s_n) \leftarrow \mathsf{Share}(s)] = 1$$

2. $(t-1)$**-security**: any subset of *less than* t parties cannot compute s, i.e., for all secrets $s \neq s'$ and for all sets $J \subset \{1, ..., n\}$ with $\mid J \mid < t$ it holds that

$$\{\{s_j\}_{j \in J} \mid (s_1, ..., s_n) \leftarrow \mathsf{Share}(s)\} \approx \{\{s_j\}_{j \in J} \mid (s_1, ..., s_n) \leftarrow \mathsf{Share}(s')\}$$

In other words, we are stating that for any set J with cardinality less than t, when examining the shares generated by the Share algorithm, if we take a different secret s', the shares are distributed in the same way. Therefore, as long as we observe less than t (the threshold) shares, we cannot distinguish whether they originated from secret s or s'. This property is a kind of indistinguishability.

In summary, the name threshold secret sharing comes from the fact that t many of these n parties together can reconstruct the secret s, however, the Share algorithm reveals nothing about the secret unless a sufficient number of shares are obtained. The $\mathsf{Reconstruct}$ algorithm in the definition takes as input t shares. We use a double index i_k because the shares used for reconstruction may not be provided in any particular order, any combination of t shares should be sufficient to reconstruct the secret.

1.2 Additive Secret Sharing

As a first example, consider an n-out-of-n secret sharing scheme, where the secret s is split into n shares distributed among n parties. In this case, all n shares are required to reconstruct the secret.

Example 1. Additive n-out-of-n Secret Sharing: Let N be a large integer (publicly known) and $s \in \mathbb{Z}_N$ be the secret value to be shared. Then the Share and $\mathsf{Reconstruct}$ algorithms work as follows:

- $\mathsf{Share}(s)$: $s_1, \ldots, s_{n-1} \xleftarrow{\$} \mathbb{Z}_N$, $s_n = s - (s_1 + \ldots + s_{n-1}) \mod N$, return $s_1, \ldots, s_n \in \mathbb{Z}_N$;
- $\mathsf{Reconstruct}(s_1, \ldots, s_n)$: compute $s = s_1 + \cdots + s_n \mod N$

The algorithms are efficiently computable. Also, it satisfies both $t-$correctness since any set of $t(= n)$ parties can reconstruct the secret s (together) and $(t-1)-$security since any subset of less than n parties cannot compute the secret s.

1.3 Replicated Secret Sharing

In the previous example, all the shares were required for reconstruction. Now, we are going to develop a 2-out-of-3 scheme. The name *Replicated* comes from the fact that some elements are replicated across different share holders.

Example 2. 2-out-of-3 Secret Sharing, Replicated Secret Sharing: Fix $n = 3, t = 2, s \in Z_N$,

- $\mathsf{Share}(s)$: pick at random $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_N$, compute $x_3 = s - x_1 - x_2 \mod N$, create the shares as set

$$s_1 = \{(2, x_2), (3, x_3)\} \qquad s_2 = \{(1, x_1), (3, x_3)\} \qquad s_3 = \{(1, x_1), (2, x_2)\}$$

return $s_1, s_2, s_3 \in (\{1, 2, 3\} \times \mathbb{Z}_N)^2$

- $\mathsf{Reconstruct}(s_i, s_j)$: compute s as the sum of all elements in the set $s_i \cup s_j$ modulo N.

Note that in replicated secret sharing, every piece of information s_i is built by an index j and a value x_j. The algorithms are efficiently computable and satisfies $2-$correctness since any set of 2 parties can reconstruct the secret s (together), indeed

$\rightarrow$ from s_1, s_2, we get $s_1 \cup s_2 = \{(2, x_2), (3, x_3), (1, x_1)\}$, hence $s = x_1 + x_2 + x_3 \mod N$;

$\rightarrow$ from s_1, s_2, we get $s_2 \cup s_3 = \{(1, x_1), (3, x_3), (2, x_2)\}$, hence $s = x_1 + x_2 + x_3 \mod N$;

$\rightarrow$ from s_1, s_2, we get $s_1 \cup s_3 = \{(2, x_2), (3, x_3), (1, x_1)\}$, hence $s = x_1 + x_2 + x_3 \mod N$;

The scheme is also $1-$secure, since any subset of less than 2 parties cannot compute the secret s.

The issue with this approach is that it is not optimal: to share a single value in $\mathbb{Z}_N$, each party must store two values in $\mathbb{Z}_N$ along with their respective index.

1.4 Shamir t-out-of-n Secret Sharing

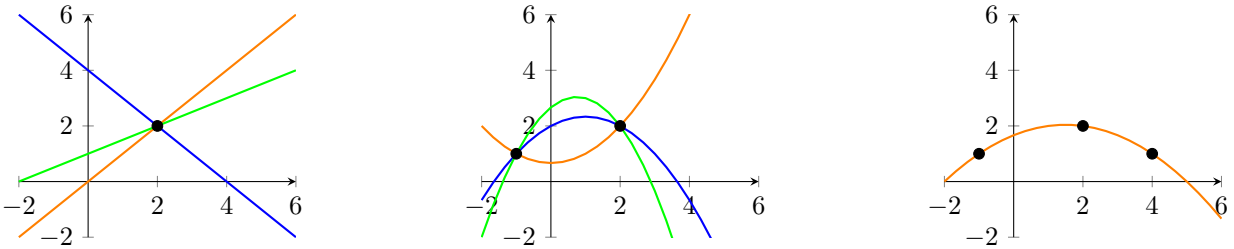
We will now see a construction of a threshold secret sharing scheme that is optimal in terms of share size, and information theoretic secure. This construction is named after the cryptographer and inventor Adi Shamir who first came up with it (Shamir, is also the S that contributed to the design of the RSA cryptosystem). Before presenting the construction, we recall a few facts on polynomials that will be fundamental to understand its correctness and security.

1.4.1 Facts on Polynomials

Theorem 1. Recall a few facts on polynomials:

1. A degree d polynomial $f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_0 \in \mathbb{Z}_N[x]$ with $a_d \neq 0$ is completely defined by $d + 1$ values in $\mathbb{Z}_N$.
2. Given d points, there are many polynomials of degree d that pass through that set of points.
3. Given $d + 1$ points, there is only one polynomial of degree d that passes through those $d + 1$ points.

We shall not provide a formal proof of the theorem; however, the accompanying figures offer an intuitive understanding of its meaning. Referring to Figure 5a, we observe the following: through a single point, there are infinitely many lines (degree-1 polynomials) that pass through it (when working over the finite and discrete plane $\mathbb{Z}_N \times \mathbb{Z}_N$ there are only finitely many lines, however, multiple lines pass through one point). Similarly, Figure 5b shows that given two points, there are (infinitely) many parabolas, i.e., degree-2 polynomials, that pass through those two points. In the final illustration Figure 5c, it is evident that through three points, there exists only one unique parabola (a degree-2 polynomial) that passes precisely through all three points. And similarly, given 2 points, there is only one line that passes through those points (on an Euclidean plane).



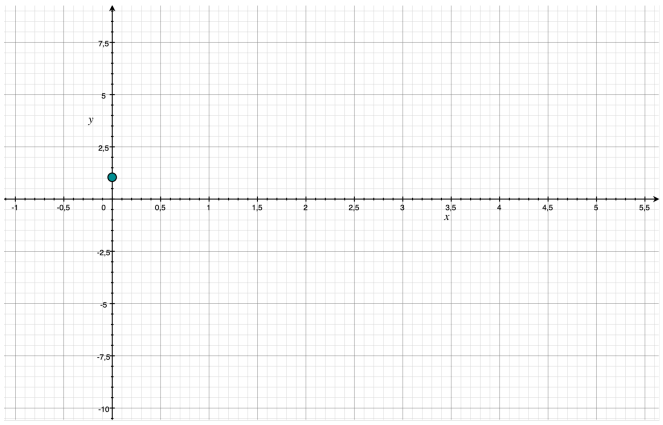
(a) (b) (c)

Figure 5: Plot of polynomials

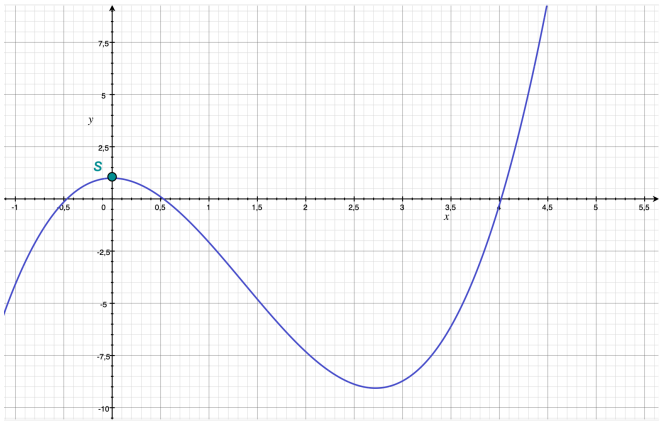
1.5 Shamir's secret sharing from polynomials

Shamir's secret sharing scheme uses polynomials to split the shares. Intuitively it works by constructing a polynomial of degree $d = t - 1$ so that if an adversary gets a hold of $t - 1$ shares, there are an several possible solutions of polynomials that pass through the set of stolen shares. Since each polynomial is different, it will reconstruct a different share, hence the original secret remains hidden. However, collecting t shares, these uniquely determine a polynomial of degree $t - 1$ (that passes through all t shares) and this polynomial will yield the original secret.

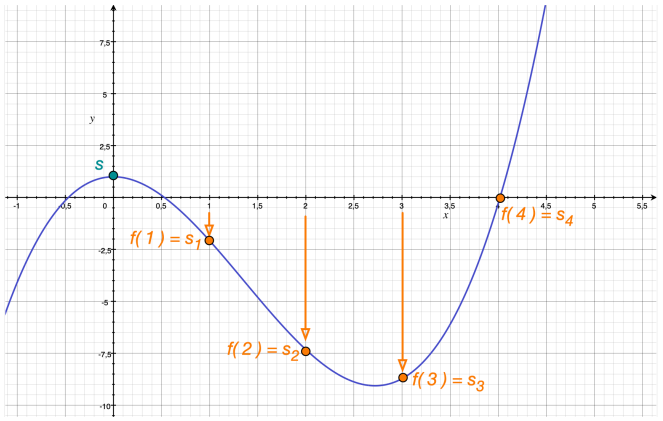
Next, we consider an intuitive step-by step example about how to compute such shares for a degree 3 polynomial:



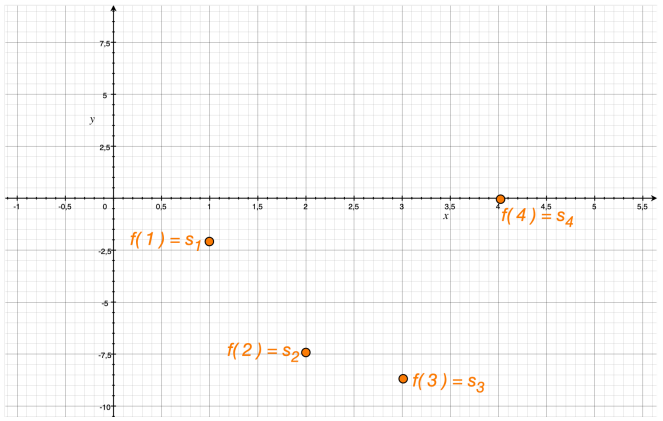
Let $s = 1$ be the secret value, the green point in the picture on the y axis.



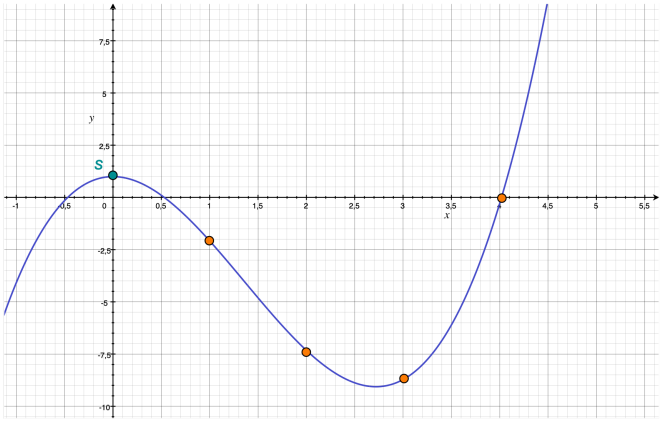
Construct a random polynomial of degree $t - 1$ passing by the point $(0, s)$ for example the blue line plots in the graph pf equation $y = (x^2 + 0.001)(x - 4.08) + 1$.



Compute the share for party P_i as $s_i = f(i) \mod p$ for $i \in \{1, 2, 3, 4\}$



If we remove the polynomial, the shares s_i look completely random, they are points on the cartesian plane.



The points s_i look completely at random unless we interpolate these $t = 4$ shares, then there exists only one polynomial of degree $3 = t-1$ that passes by all of them. The polynomial is exactly f, and $f(0) = s$ is exactly the secret (reconstructed by interpolation from the shares).

Definition 2. Shamir t-out-of-n Secret Sharing:

- $\mathsf{Share}(s)$: given the value $s \in \mathbb{Z}_p$ (where p is a prime or a power of a prime) do:
  - Sample $t-1$ random values $a_1, ..., a_{t-1} \xleftarrow{\$} \mathbb{Z}_p$, with $a_{t-1} \neq 0$.
  - Construct the polynomial $f(x) = s + a_1 x + a_2 x^2 + ... + a_{t-1} x^{t-1} \in \mathbb{Z}_p[x]$.
  - Compute the n shares by evaluating $f(x)$ on n distinct points: $s_i = f(i)$ for $i \in \{1, ..., n\}$.
  - Send the shares s_i to party P_i (via a secure channel).
- $\mathsf{Reconstruct}(s_{i_1}, s_{i_2}, ..., s_{i_t})$: let $I = \{i_1, ..., i_t\}$ be the set distinct share indexes do:
  - Compute the Lagrange coefficients for the set I: $\ell_i^I(0) = \prod_{j \in I \setminus \{i\}} \frac{j}{j-i} \mod p$
  - Recover the secret $s = \sum_{i \in I} s_i \cdot \ell_i^I(0) \mod p$

The algorithms are efficiently computable and fulfill t-**correctness** and $(t-1)$-**security**.

Correctness The correctness of this scheme follows by the following theorem:

Theorem 2 (Polynomial Identity)**.** $f(x)$ and $\sum_{i \in I} f(i)\ell_i^I(x)$ coindice on $d+1$ points where

$$\ell_i^I(x) := \prod_{j \in I \setminus \{i\}} \frac{j-x}{j-i} \mod p$$

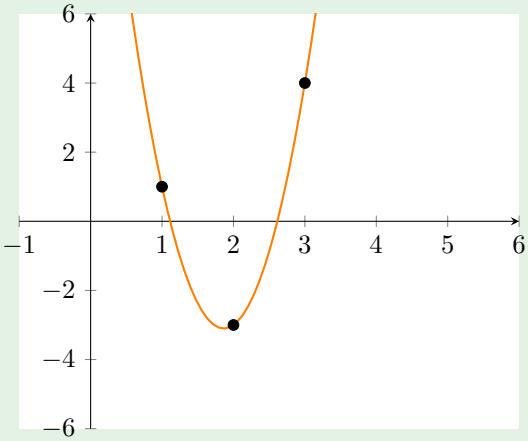
are the Lagrange coefficients for the set I.

Theorem 2 states that $f(x)$ and $\sum_{i \in I} f(i)\ell_i^I(x)$ coincide at $d+1$ distinct points. Since $f(x)$ is a polynomial of degree d, it follows that $f(x)$ and $\sum_{i \in I} f(i)\ell_i^I(x)$ must be identical polynomials.

Note that the polynomials $\ell_i^I(x)$ depend solely on $i, j \in I$, and not on $f(x)$, thus these polynomials are independent of the secret. Furthermore, the polynomials $\ell_i^I(x)$ exist for every i and every set $I \subset \mathbb{Z}_p$ of cardinality t. By setting $x = 0$, we retrieve the secret s, indeed

$$s = f(0) = f(i_1)\ell_{i_1}^I(0) + f(i_2)\ell_{i_2}^I(0) + \ldots + f(i_t)\ell_{i_t}^I(0) \mod p$$

Example 3. By selecting three fixed points of the second degree polynomial $f(x)$, there exists a unique degree-2 polynomial (parabola) that passes through these points, as illustrated in the graph.



Let $g(x)$ the funciton define as

$$\begin{aligned} g(x) &:= f(1)l_1^I(x) + f(2)l_2^I(x) + f(3)l_3^I(x) \mod p = \\ &:= f(1) \cdot \frac{2-x}{2-1} \cdot \frac{3-x}{3-1} + f(2) \cdot \frac{1-x}{1-2} \cdot \frac{3-x}{3-2} + f(3) \cdot \frac{1-x}{1-3} \cdot \frac{2-x}{2-3} \mod p = \\ &:= f(1) \cdot \frac{2-x}{1} \cdot \frac{3-x}{2} + f(2) \cdot \frac{1-x}{-1} \cdot \frac{3-x}{1} + f(3) \cdot \frac{1-x}{-2} \cdot \frac{2-x}{-1} \mod p \end{aligned}$$

hence we get the polynomials coincide on 3 points

$$g(i) = f(i)$$

for each $i \in \{1, 2, 3\}$

$$\begin{cases} g(1) = f(1) \cdot \frac{2-1}{1} \cdot \frac{3-1}{2} + f(2) \cdot \frac{1-1}{-1} \cdot \frac{3-1}{1} + f(3) \cdot \frac{1-1}{-2} \cdot \frac{2-1}{-1} \mod p = f(1) \\ g(2) = f(1) \cdot \frac{2-2}{1} \cdot \frac{3-2}{2} + f(2) \cdot \frac{1-2}{-1} \cdot \frac{3-2}{1} + f(3) \cdot \frac{1-2}{-2} \cdot \frac{2-2}{-1} \mod p = f(2) \\ g(3) = f(1) \cdot \frac{2-3}{1} \cdot \frac{3-3}{2} + f(2) \cdot \frac{1-3}{-1} \cdot \frac{3-3}{1} + f(3) \cdot \frac{1-3}{-2} \cdot \frac{2-3}{-1} \mod p = f(3) \end{cases}$$

Since $g(x)$ and $f(x)$ are equal at three distinct points, it follows from the theorem that $g(x)$ and $f(x)$ are the same polynomial.

This algorithm is efficiently computable, as it involves only addition and multiplication operations. The correctness follows from the fact that, given t points, there exists a unique polynomial of degree $t-1$ that passes through these points. On the other hand, the security of the scheme arises from the fact that, given fewer than t points, there are infinitely many polynomials of degree $t-1$ that can pass through those points, making it impossible to deduce the secret from fewer than t shares.

1.6 Threshold Cryptography

The new tool, Secret Sharing, can now be combined with previously presented tools to achieve *Threshold Cryptography*: this captures scenarios where security depends on multiple entities or secrets and aim to address adversaries that can compromise some, but not all, of them.

Example 4. Threshold El-Gamal's Cryptosystem. Let be p a large prime and g an element of $\mathbb{Z}_p^*$ which generates a cyclic (multiplicative) subgroup of prime order q.

- $KeyGen(1^n)$ is the same setting as ElGamal: sample $x \xleftarrow{\$} \mathbb{Z}_q^*$, set $\mathsf{pk} = g^x$, use Shamir secret sharing to compute $(\mathsf{sk}_1, ..., \mathsf{sk}_n) \leftarrow \mathsf{Share}(x)$. Output pk and send sk_i to party i.
- $Enc(\mathsf{pk}, m)$: sample $r \xleftarrow{\$} \mathbb{Z}_q^*$, output the ciphertext $c = (c_1 = g^r, c_2 = m \cdot \mathsf{pk}^r)$.
- $Parital.Dec(\mathsf{sk}_i, c)$: parse $c = (c_1, c_2)$, compute $d_i = c_1^{\mathsf{sk}_i}$.
- $Collaborate.Dec(I, \{d_i\}_{i \in I}, c_2)$: compute $D = \prod_{i \in I} d_i^{\ell_i^I(0)}$, output $m = c_2 \cdot D^{-1}$.

The key generation $KeyGen$ and the encryption Enc algorithm are essentially the same as the *classical* El-Gamal. The distinction lies in the decryption, which consists of two algorithms: a partial decryption algorithm $Parital.Dec$ and a collaborative decryption algorithm $Collaborate.Dec$. This is because no one possesses the entire secret key, but only a shares of it, thus can only perform a partial decryption. And therefore has to collaboratively, using the decryption algorithm $Collaborate.Dec$, combine the partial decrypted values to reconstruct the plaintext.

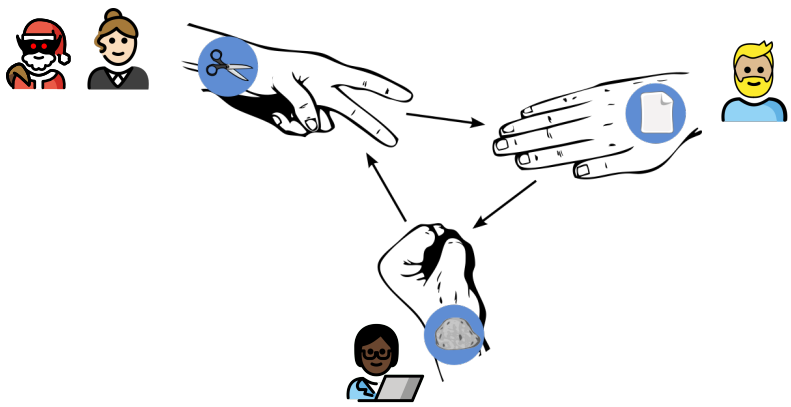
The correctness of the previous scheme follows from:

$$\prod_{i \in I} d_i^{\ell_i^I(0)} = \prod_{i \in I} \left(c_1^{\mathsf{sk}_i}\right)^{l_i} = \prod_{i \in I} \left((g^r)^{\mathsf{sk}_i}\right)^{\ell_i^I(0)} = \prod_{i \in I} (g^r)^{\mathsf{sk}_i \cdot \ell_i^I(0)} = (g^r)^{\sum_{i \in I} \mathsf{sk}_i \cdot \ell_i^I(0)} = (g^r)^x = g^{rx}$$

hence : $c_2 \cdot D^{-1} = (m \cdot \mathsf{pk}^r) \cdot (g^{-rx}) = m \cdot g^{rx} \cdot g^{-rx} = m$.

2 Commitment Schemes

To give some intuition, consider the game of Rock-Paper-Scissors. An important part of this game is to coordinate the exact moment when everyone shows their hands, this is *easy* to do in the game, but much more difficult online. Thus for this we use commitment scheme, which is a way of committing to a choice and revealing it later. Next we formalize this and define the security requirements.



Definition 3. A **cryptographic commitment function** is a deterministic polynomial time algorithm $\mathsf{Commit} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^*$ that takes in input a message m and a random string $r \in \{0,1\}^n$ (here n denotes the security parameter); and outputs a commitment c to the message m.

Definition 4. A **commitment scheme** is an interactive protocol between two parties (a sender S and a receiver R). It uses a cryptographic commitment function and must satisfy the **binding** and **hiding** properties given later. It is composed of two phases:

1. **Commit Phase.** The sender S commits to a value m by computing $c = \mathsf{Commit}(m, r)$ for a random value $r \xleftarrow{\$} \{0,1\}^n$, and sends c to the receiver R.
2. **Reveal Phase.** The sender S 'opens' its commitment by sending (m, r) to the receiver. The receiver R verifies that the value c it got during the commit phase matches $\mathsf{Commit}(m, r)$ for the values (m, r) it got during the reveal phase.

2.1 Security Properties: Hiding & Binding

For commitment schemes we consider two security features: **Hiding** and **Binding**. These properties can be complexity-based or unconditional, this is referred to as *computational* resp. *information-theoretic*. In the case of computational, then $\mathcal{A}$ is a PPT algorithm. In the information-theoretic case, there are no such restrictions on the adversary's power, running time, nor computational abilities.

Informally speaking, binding protects the receiver's security, since it ensures the sender cannot change their mind after sending the commitment. On the other hand, hiding protects the sender, ensuring that the receiver cannot learn the committed value before receiving the opening of the commitment.

Definition 5. A commitment scheme is said to be **binding** if no adversary $\mathcal{A}$ can find two distinct messages $m \neq m^*$ (and randomness $r, r^* \in \{0,1\}^n$) that yield the same commitment value: $c = \mathsf{Commit}(m, r) = \mathsf{Commit}(m^*, r^*)$. Formally, for bounded adversaries:

$$Pr[\mathsf{Commit}(m, r) = \mathsf{Commit}(m^*, r^*) | m \neq m^* \wedge ((m, r), (m^*, r^*)) \leftarrow \mathcal{A}] \leq negl(n).$$

While for unbounded adversaries (information theoretic):

$$Pr[\mathsf{Commit}(m, r) = \mathsf{Commit}(m^*, r^*) | m \neq m^* \wedge ((m, r), (m^*, r^*)) \leftarrow \mathcal{A}] = 0.$$

A commitment scheme is called **computationally binding** if the adversary $\mathcal{A}$ is a PPT algorithm and the probability is negligible. If no such restriction is made the scheme is called **information-theoretically binding**.

Definition 6. A commitment scheme is said to be **hiding** if no adversary $\mathcal{A}$ can win the following game with probability noticeably higher than $\frac{1}{2}$:

1. The adversary $\mathcal{A}$ outputs two messages m_0 and m_1.
2. The challenger $\mathcal{C}$ selects a random bit $b \xleftarrow{\$} \{0,1\}$; picks a random $r \xleftarrow{\$} \{0,1\}^n$; computes $c = \mathsf{Commit}(m_b, r)$; and returns c to $\mathcal{A}$.
3. the adversary outputs a bit b^* as a guess for b.

The adversary $\mathcal{A}$ is said to win the game if $b^* = b$, else $\mathcal{A}$ looses the game.
Formally, for bounded adversaries:

$$Pr\left[b = b^* \,\middle|\, \begin{array}{c} (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{Commit}) \\ b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \mathbb{Z}_q \\ \mathsf{Commit}(m_b, r) = c \\ b^* \leftarrow \mathcal{A}(c) \end{array}\right] = \frac{1}{2} + negl(n)$$

For unbounded adversaries (information theoretic hiding) $negl(n)$ is replaced by $\frac{1}{2^n}$.

A commitment scheme is called **computationally hiding** if $\mathcal{A}$ is a PPT algorithm and $Pr[b = b^*] \leq \frac{1}{2} + negl(n)$.

If no such restriction is made, the scheme is called **information theoretically hiding** and $Pr[b = b^*] = \frac{1}{2} + 2^{-n}$.

Remark 1: most cryptography is concerned with computational security, which is based on the belief that certain problems are computationally difficult to solve. Additionally, there is information-theoretic security, which asserts that, regardless of the computational resources available, the problem remains unsolvable.

Remark 2: The hiding property resembles the IND-CPA security game of encryption schemes. A natural question that rises is then: is it possible realize a Commitment Scheme using any Encryption Scheme? A secure encryption scheme is definitely hiding (it is at least IND-CPA secure), but it is not always binding. For example, the One Time Pad is binding. If $m \oplus r = c$, then $\mathcal{A}$ can break binding by providing an alternative opening $m^* \neq m$ and $r^* = c \oplus m^*$.

2.1.1 Impossibility Result

Theorem 3. No commitment scheme can be information-theoretically binding *and* information-theoretically hiding at the same time.

Proof. (by reduction to absurd) Suppose we have a scheme which is both information-theoretically hiding and binding, and suppose the committing party (the sender) generates a commitment $c = \mathsf{Commit}(m, r)$. The information-theoretical hiding property implies that there must exist values $m^*(\neq m)$ and some randomness r^* such that $c = \mathsf{Commit}(m^*, r^*)$. This is because otherwise, an infinitely powerful receiver could break the hiding property (that is finding the unique pair (m, r) that generates the commitment value c).

Thereby the information-theoretical hiding property implies that there exists an m^* s.t. $\mathsf{Commit}(m^*, r^*) = c = \mathsf{Commit}(m, r)$ which means the commitment is not binding (and an infinitely powerful sender can find such a collision m^*).

We conclude that if the commitment scheme is information-theoretically hiding it cannot be binding for a computationally unbounded sender, which contradicts the statement of the theorem. The conclusion is that it was absurd to assume the existence of a commitment scheme that is both hiding and binding in an information-theoretically way.

□

2.2 Hash-Based Commitments

What happens if we have a simple construction where the commitment function is a hash function $c = H(m)$? It would be binding, but not hiding as one can easily see which choice was made even though it is hashed. But, if one concatenates one choice together with some randomness before hashing, then it will be both binding and hiding:

$$\mathsf{Commit}(m, r) = H(m||r) = c$$

When everyone has committed, we can reveal the randomness. In more detail we can see that this scheme is both hiding and binding:

- **Binding** by the second preimage resistance of the hash function H. If you could change the message, it would mean that you have found two different messages $(m||r)$ and $(m^*||r^*)$ that produce the same hash. However, we know that finding a second-preimage collision is supposed to be computationally hard:

$$Pr\left[H(m||r) = H(m^*||r^*) \;\middle|\; m \neq m^*\right] \leq negl(|r|)$$

- **Hiding** by preimage resistance of the hash function H. If we have sufficiently long randomness, the probability of determining whether the hash comes from m_0 or m_1 is negligible since we know that the hash function H is preimage resistance

$$Pr\left[b^* = b \;\middle|\; m_0, m_1, H(m_b||r)\right] \leq negl(|r|)$$

2.3 Pedersen Commitment

Definition 7. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q, (inside the multiplicative group $\mathbb{Z}_p^*$). Let h be a random element in $\mathbb{G} \setminus g$. Let p, q, g, h all be public information. **Pedersen commitment functions** is defined as:

$$\mathsf{Commit}(m, r) = g^m h^r (\mod p)$$

for $r \xleftarrow{\$} \mathbb{Z}_q, m \in \mathbb{Z}_q$.

Theorem 4. Pedersen Commitment scheme is computationally binding and information-theoretically hiding.

Proof.

Computationally Binding: Pedersen Commitment scheme is computationally binding, as it reduces to DL, as it is shown in the Figure 6.

If $\mathcal{A}_{\text{Binding}}$ breaks binding, meaning that, given the input (g, h), the adversary $\mathcal{A}_{\text{Binding}}$ provides $((m, r), (m^*, r^*))$ that yield the same commitment value. The reduction $\mathcal{A}'$ solves the DL instance $(g, h = g^x)$ by sending $x = (m^* - m)(r - r^*)^{-1} \mod q$.

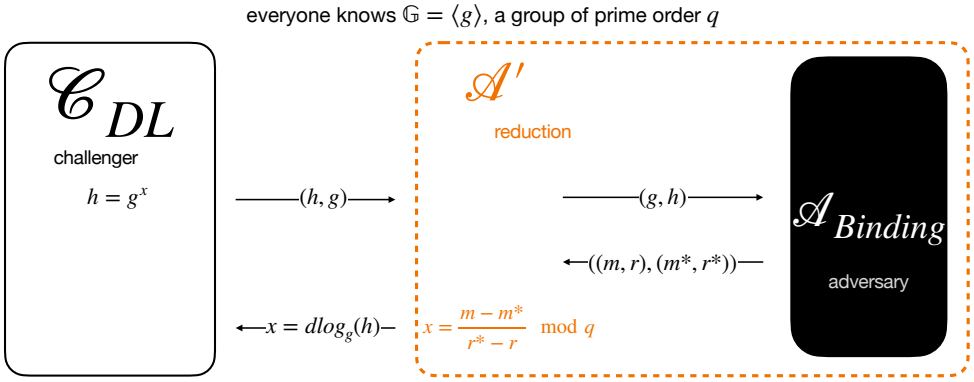


Figure 6

In details, this works because we have

$$\mathsf{Commit}(m, r) = c = \mathsf{Commit}(m^*, r^*)$$

with $m \neq m^*$, hence we can derive the discrete logarithm

$$\begin{aligned}
\mathsf{Commit}(m, r) &= \mathsf{Commit}(m^*, r^*) \\
g^m \cdot h^r \mod p &= g^{m^*} \cdot h^{r^*} \mod p \\
g^m \cdot g^{-m^*} \mod p &= h^{-r} \cdot h^{r^*} \mod p \\
g^{m-m^*} \mod p &= h^{-r+r^*} \mod p \\
\left(g^{m-m^*}\right)^{(-r+r^*)^{-1}} \mod p &= \left(h^{-r+r^*}\right)^{(-r+r^*)^{-1}} \mod p \\
g^{(m-m^*)\cdot(-r+r^*)^{-1}} \mod p &= h \mod p
\end{aligned}$$

Remark: since $m \neq m^*$, we know also that $r \neq r^*$.
In summary, we have just proven that if such an adversary $\mathcal{A}_{\text{Binding}}$ exists, it would be possible to extract the discrete logarithm DL. However, since solving the discrete logarithm problem is computationally hard, such an adversary $\mathcal{A}_{\text{Binding}}$ cannot exist.

Information-Theoretically Hiding Pedersen Commitment scheme is information-theoretically hiding. This can be proven by showing that for any commitment c, there exists r_0, r_1 such that $c = g^{m_0} h^{r_0} = g^{m_1} h^{r_1}$, similarly to OTP.
In detail we have that the advantage of guess is the message is equal to 1 over the size of the randomness which, in this case, gives $\frac{1}{q}$: given a commitment c, since for every message $m \in \mathbb{Z}_q$, we can find a randomness r that can explain it:

$$\begin{aligned}
\mathsf{Commit}(m_0, r_0) &= c = \mathsf{Commit}(m_1, r_1) \\
g^{m_0} \cdot h^{r_0} &= c = g^{m_1} \cdot h^{r_1} \\
g^{m_0} \cdot g^{x \cdot r_0} &= c = g^{m_1} \cdot g^{x \cdot r_1}
\end{aligned}$$

$$\begin{cases} r_0 = (c - m_0) \cdot x^{-1} \\ r_1 = (c - m_1) \cdot x^{-1} \end{cases}$$

□

TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Applying cryptography to protect privacy

Lecturer: Victor Morel Lecture 10 on Dec 6

Scribes: Willem Brahmstaedt, Hanna Ek & Lucia Lavagnino Last update December 30, 2024

Contents

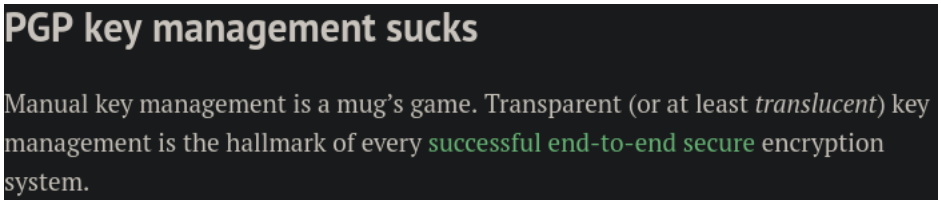
1 Usability and uptake of Privacy Enhancing Technologies

In 2013, Edward Snowden[1] disclosed that the National Security Agency (NSA) in the United States of America was engaged in global surveillance. In a famous article[2] he revealed that the British intelligence agency GCHQ had been tapping into fiber optic cables to intercept worldwide communications. In another article[3] , Snowden exposed that the NSA was primarily focused on analyzing metadata rather than the content of communications themselves. This meant that the agency was not directly monitoring the substance of conversations, but rather identifying the parties involved and their patterns of communication. These revelations significantly raised awareness about privacy issues, prompting widespread debate both in academia and across the globe, as people began to pay closer attention to the implications of privacy in the digital age.

How can we design tools to protect user's right to privacy and secrecy so that the tools are truly usable, and do not turn into accidental traps, or "only for geeks" gadgets? This lecture aims to examine the usability of cryptographic tools and conduct an in-depth analysis of a specific Privacy-Enhancing Technology (PET). Furthermore, we will broaden our perspective to explore the role of cryptography at the societal level. How can secure communication be achieved? What are the implications of usability and uptake in this context?

1.1 Securing communications

PGP, which stands for Pretty Good Privacy, is a cryptographic system designed to encrypt emails. Developed 30 years ago, it remains in use today. The cryptography behind PGP is highly secure and has yet to be compromised. However, its complexity makes it difficult to use, contributing to its limited adoption.



Another example can be found on the web: when we connect securely to a website, we use *HTTPS*, where the *"S"* stands for *"Secure"*. To establish this secure connection, a certificate is required to verify the legitimate

[1] For more detail see https://it.wikipedia.org/wiki/Edward_Snowden
[2] https://www.theguardian.com/uk/2013/jun/21/gchq-cables-secret-world-communications-nsa
[3] https://www.theguardian.com/world/2013/jun/08/nsa-boundless-informant-global-datamining

ownership of the website and ensure that it is not a phishing site. This process relies on the TLS (Transport Layer Security) protocol to facilitate secure communication.
A few years ago, purchasing a certificate was a costly endeavour. For instance, in 2002, GeoTrust sold its certificates for $119 per year per website, while VeriSign's certificates ranged between $250 and $350. As a result, encryption was not widely adopted, and only a limited number of websites were encrypted. This lack of widespread encryption allowed agencies like the NSA to access and monitor metadata more easily. Today, we have end-to-end encryption (E2E) implemented in platforms such as Signal, WhatsApp, and Matrix, which ensures that message communications are securely encrypted from sender to recipient. Additionally, services like ProtonMail and Tutanota provide encrypted email solutions, safeguarding the confidentiality of email communication. Since these technologies are widely available, people are using them and as a result, communication is now secure by default.
Referring back to the communications disclosed by Snowden, now there are secure methods for whistleblowing, such as "Guarding Securely"[4]. A whistleblower is an individual who discloses information that, while not necessarily illegal, it is morally important. This platform offers guidelines to help individuals contact them securely. They provide usable solutions for secure communication through cryptography, tailored to various threat levels. Additionally, they outline the advantages and disadvantages of each solution to ensure informed decision-making.
In 2014, Google changed its rules, making HTTPS almost mandatory (incentive via SEO): websites that did not implement HTTPS saw a lower ranking in search results. This shift, along with initiatives like *Let's Encrypt* and *HTTPS Everywhere*, played a crucial role in making encryption more accessible and affordable. The availability of free certificates and the widespread adoption of HTTPS has significantly increased the security of web traffic. Today, HTTPS is ubiquitous, with more than 95% of global web traffic is now encrypted.

1.2 Securing Data and Systems

In 2022, a data breach occurred[5] , and Professor Andrei Sabelfeld (from Chalmers University) was interviewed following the incident. The breach involved a schooling system that was leaking sensitive information, such as names, emails, and passwords. Often, such leaks happen because organizations continue to rely on outdated cryptographic methods.
A more well-known example involves the use of the MD5 hash function[6] –a deprecated cryptographic algorithm due to its high number of vulnerabilities. MD5 can be broken in a matter of minutes using a standard laptop, highlighting the risks of relying on outdated cryptographic algorithms.

One way to protect passwords on devices is by using a password manager. A password manager is a technology tool that helps internet users create, save, manage and use passwords across different online services.
Using a password manager is a good security practice, but it is important to acknowledge that password managers themselves can sometimes be compromised. For instance, *LastPass*, a well-known password manager, was breached in the past. However, there are reliable and secure alternatives, such as *Bitwarden*, which offer strong encryption and a good track record in safeguarding user data.

2 Attribute Based Credentials (ABC)

To illustrate this concept, consider Systembolaget in Sweden, which allows the purchase of alcohol only for individuals over the age of 20. If Alice wishes to prove her age, she could present her ID. However, the ID contains not only her age but also other sensitive and unnecessary information, such as her exact date of birth, nationality, gender, and name, even though only her age (over 20) is required.
One solution to this issue is using of Attribute-Based Credentials (ABC). With ABC, Alice can prove that she is over 20 without disclosing her exact age or any other personal details. The system allows her to present only the relevant attribute (her age), while keeping other private information concealed, thus ensuring both privacy and the necessary proof of eligibility.
Attribute-Based Credentials are a form of authentication mechanism that enables the flexible and selective verification of various attributes of an entity, while ensuring that no additional information about the entity is disclosed (by the zero-knowledge property). For instance, about the previous scenario, Alice can demonstrate to Systembolaget that she is over 20 without disclosing her exact age.
ABC is a system that requires some basic blocks and properties. Additionally, there are key requirements that the protocol must satisfy.

2.1 Basic blocks

ABC protocols combine fundamental components such as Pedersen commitments, group signatures, blind signatures, and zero-knowledge proofs (ZKPs). ABC can be realised as a (generalised) Pedersen commitment scheme.

[4] https://www.theguardian.com/help/ng-interactive/2017/mar/17/contact-the-guardian-securely
[5] https://www.svt.se/nyheter/lokalt/vast/it-experten-om-vklass-och-lackta-elevuppgifterna-i-goteborg
[6] https://cybernews.com/security/xado-leaks-us-phone-numbers-emails-md5-unsalted-passwords/

Brands [7] proposes a signature scheme that can be utilized for Attribute-Based Credentials. Camenisch and Lysyanskaya [8] propose an alternative signature scheme for constructing Attribute-Based Credentials, which is also based on a Pedersen commitment.

A group signature scheme is a method for allowing a member of a group to anonymously sign a message on behalf of the group.

For example, a group signature scheme could be used by an employee of a large company where it is sufficient for a verifier to know a message was signed by an employee, but not which particular employee signed it.

Essential to a group signature scheme is a group manager, who is in charge of adding group members and has the ability to reveal the original signer in the event of disputes.

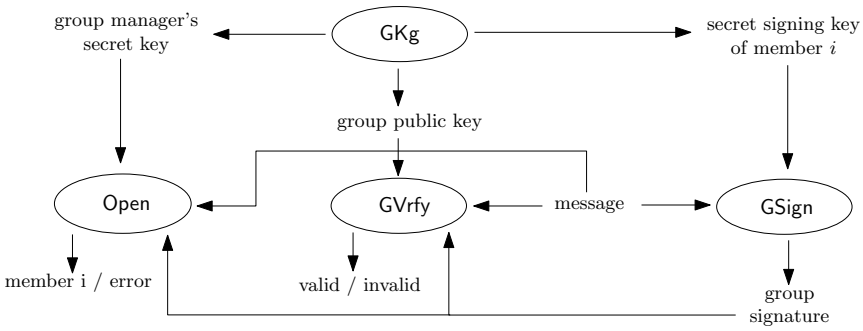


Figure 1: Static Group Signatures

The initialisation of group signatures will give the group manager a secret key, the group public key, and secret keys for the members which are a vector in practice in its simplest form. GSign you can sign on behalf of the group. Gvrfy you can verify if the message signed has indeed been signed by a member of the group. And opening by the group manager to see which member signed, if needed.

A Zero-Knowledge Proof (ZKP) operates as follows: one party (the prover) can demonstrate to another party (the verifier) that a given statement is true while ensuring that the prover does not disclose any additional information beyond the fact that the statement is indeed true. An (honest verifier) zero-knowledge proof of knowledge has to be:

Complete If the prover knows x, they can convince the verifier

Sound If the prover does not know x, they **cannot** convince the verifier

Zero knowledge The verifier does not learn any other information but that the prover knows x

2.2 Blind signatures

Blind signatures are used in ABC protocols, they are an interactive protocol between a receiver and a signer, which results in a valid cryptographic signature for the receiver without the signer learning the content of the message or the resulting signature. Formally:

Definition 1. A **blind signature scheme** is a signature scheme where the signing algorithm $Sign$ is replaced by an *interactive* protocol run between a signer/issuer (S) and a receiver (R). The protocol starts with R who has as input a message m, and S who has as input a secret key sk. At the end of the interaction, R obtains a signature σ on m, and S learns nothing about m or σ.

[7] S. A. Brands. "Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy", MIT press, 2000.

[8] Camenisch, Jan, and Anna Lysyanskaya. "A signature scheme with efficient protocols." International Conference on Security in Communication Networks., 2002

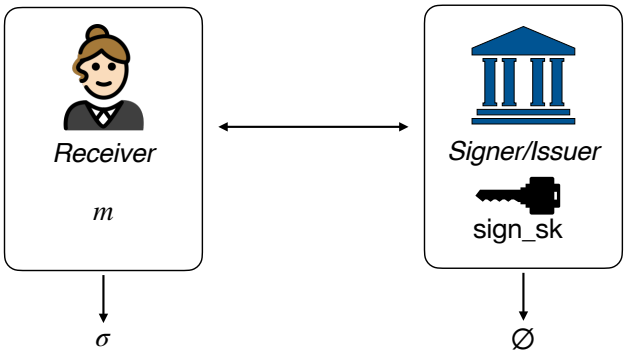


Figure 2: Idea behind bling signatures

Blind signatures are for example used in untraceable electronic payment system and attribute-based credentials.

An example of this is **Chaum's Untraceable eCash System** described in figure 3, an eCash system devised by Chaum, aka the godfather of modern cryptography. This was an early version of electronic money, similar to Bitcoin but the biggest difference compared to Bitcoin is that the banks are trusted and play a key role in the system.

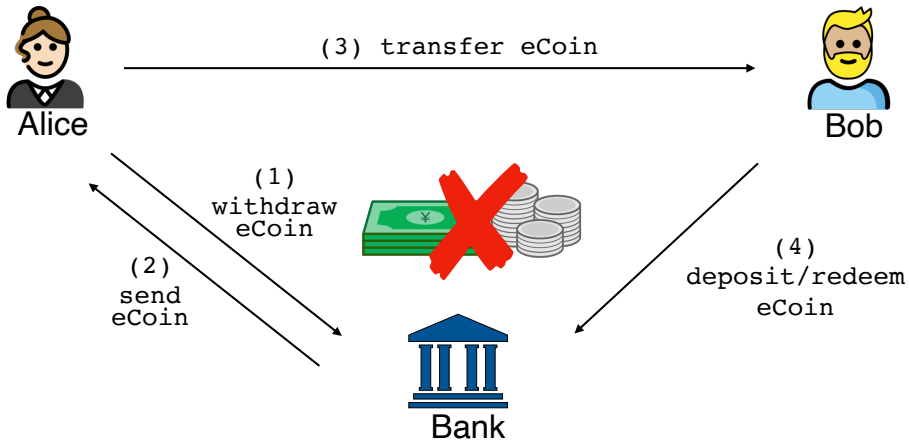


Figure 3: Chaum's Untraceable eCash System

In figure 3, there is a system that enables the withdrawal, transfer, and deposit of an electronic currency without the possibility to trace its users.
Chaum's Untraceable eCash System has three properties that need to be fulfilled.

1. Only the Bank can generate eCoins (unlike BitCoin).
2. Users cannot double-spend eCoins (money cloning). This property is also present in Bitcoin, which signifies that one cannot spend money that one does not possess.
3. eCoins should be untraceable, like physical cash. This property is stronger than bitcoin in that you cannot reconstruct the transaction graph, it is said to be unlinkable.

The solution for the first property is the following (only banks can generate eCoin): eCoin is a bit string together with a digital signature generated using the Bank's secret key (sk). Unforgeability of the signature ensures that an adversary $\mathcal{A}$, cannot generate eCoins. The solution to the second and third properties (double spending and untraceable) is blind signatures. That is we want the bank to be able to sign an eCoin without knowing what eCoin is. The procedure of a blind signature scheme RSA-FDH can be seen in Figure 4 and 5

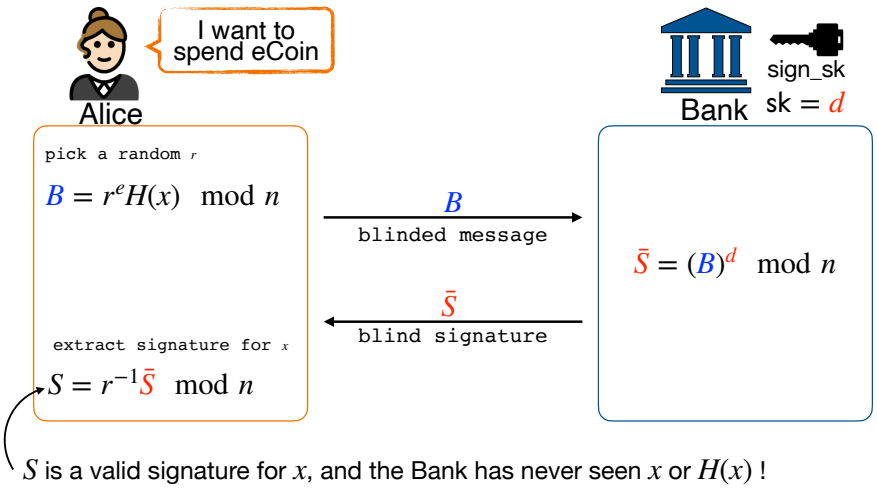


Figure 4: eCoin withdrawal procedure with RSA (blind) signatures

Let p and q be two distinct primes, $n = pq$, and $d, e \in \mathbb{Z}^*_{\varphi(n)}$ such that $d = e^{-1} \mod \varphi(n)$. As in RSA, the bank holds a secret key $\mathsf{sk} = d$, while e is the shared public key. Let r be a randomly chosen value in $\mathbb{Z}^*_n$. Alice begins by computing a blinded message $B = r^e H(x) \mod n$ for the coin x. Alice then sends the blinded message B to the bank. Upon receiving B, the bank signs it using its secret key sk, producing a blinded signature $\overline{S} = B^d \mod n$. The bank then returns the blinded signature $\overline{S}$ to Alice. To obtain a valid, usable signature, Alice computes the final signature $S = r^{-1}\overline{S} \mod n$, which she can now use for verification.
How can one check that the signature is valid? Simply by the procedure below:

$$\begin{aligned} S &= r^{-1}\hat{S} \mod n \\ S &= r^{-1}(B)^d \mod n \\ S &= r^{-1}(r^e H(x))^d \mod n \\ S &= r^{-1} \cdot r \cdot H(x)^d \mod n \\ S &= H(x)^d \mod n \end{aligned}$$

And then verified by:

$$S^e = H(x) \bmod n$$

In this way, the Bank can sign an eCoin, without knowing what eCoin it is. With this system, Alice is also able to spend the eCoin.

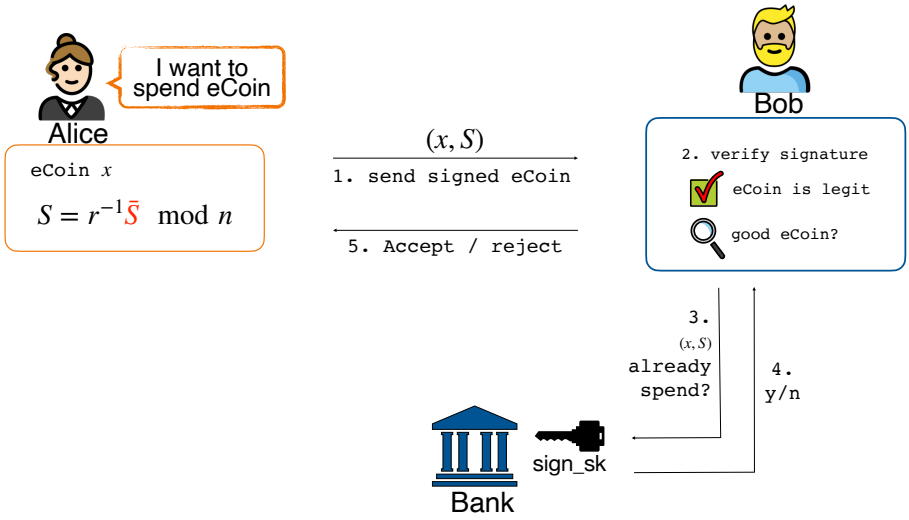


Figure 5: eCoin spending and redeeming

Alice can now make a payment to Bob by sending the coin x along with the signature S. Bob verifies the validity of the coin by checking it against the bank, which can do so because it holds the private key sk that signed the coin. Everyone can verify the validity of the coin, as the public key pk is publicly available. Moreover, Bob verifies with the bank that the coin has not been double-spent. If the eCoin has not been spent already, he accepts

the transaction.
However, there is a significant issue with this approach: while it is conceptually simple, it is not practical. The problem lies in the fact that the bank must track all transactions and maintain a record of every single one, which is an unpractical requirement.

2.3 Combination of Basic blocks

Attribute-Based Credentials can be designed as follows: first, a suitable signature scheme is constructed. This involves considering a commitment scheme, generalizing the commitment scheme to a tuple of values rather than a single value, and applying a signature on the commitment. Next, ABC protocols are developed based on the signature. This process includes using a blinded version of the signature for issuance and applying a (zero) proof of knowledge for selective disclosure.

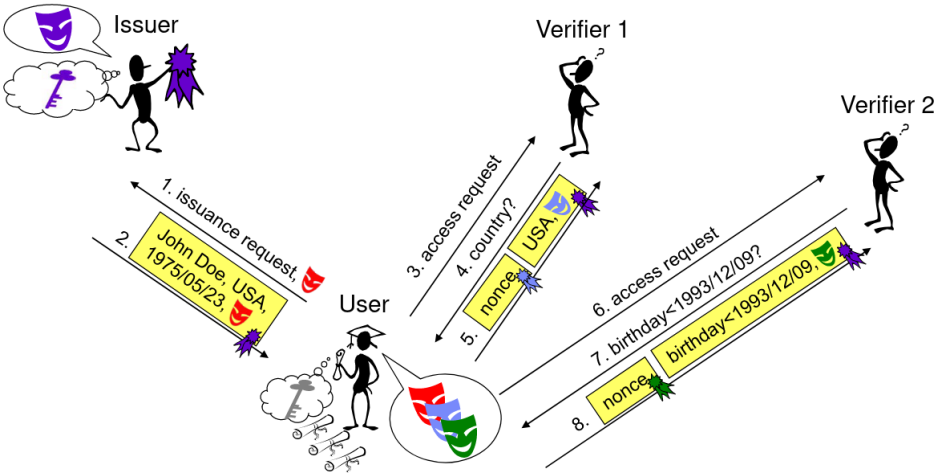


Figure 6: A possible implementation of ABC - Idemix

A possible implementation of Attribute-Based Credentials (ABC) is **Idemix**, one of the most well-known ABC systems. In the example depicted, we have a user, John Doe, who was born on May 23, 1975, in the USA. He requests a signature from the issuer. For instance, the first verifier Verifier 1 may want to confirm John's country of origin. John can then use zero-knowledge proof to disclose, in a blinded manner, the fact that he is from the USA. Since the signature has been issued by the credential issuer, it is valid, and no one can trace the authentication process. Suppose there is also a second verifier Verifier 2, who, for example, wants to verify that John is old enough. In this case, John can choose to disclose the relevant information selectively using blind signatures and zero-knowledge proofs.

For an ABC system, we also need to consider some features.
Security features (requirements)

- S1 **Authenticity**: The content of an ABC signed by the issuer cannot be modified and the verifier can verify the signature using this issuer's public key.
- S2 **Unforgeability**: A 3^{rd} party cannot forge a credential prevents a malicious third party to forge a valid ABC.
- S3 **Non-repudiation**: Credentials issuer cannot deny prevents the issuer to deny that the credential's signature was produced by him.
- S4 **Non-transferability**: A user cannot transfer his credential to another.

Privacy features (requirements)

- P1 **Offline issuer**: The issuer does not have to be online, the issuer of a credential is not involved in the verification protocol.
- P2 **Issuer unlinkability**: ABC prevents an issuer to trace his credentials. More precisely, an adversary (e.g. a colluding set of issuers and verifiers) cannot decide if an issuing protocol and a verification protocol belong to the same credential.
- P3 **Multi-show unlinkability**: Verifiers cannot trace the activities of a user. More precisely, seeing two verification protocols, no adversary (e.g. a colluding set of verifiers and issuers) can distinguish the cases whether those protocols were performed using the same credential or not.

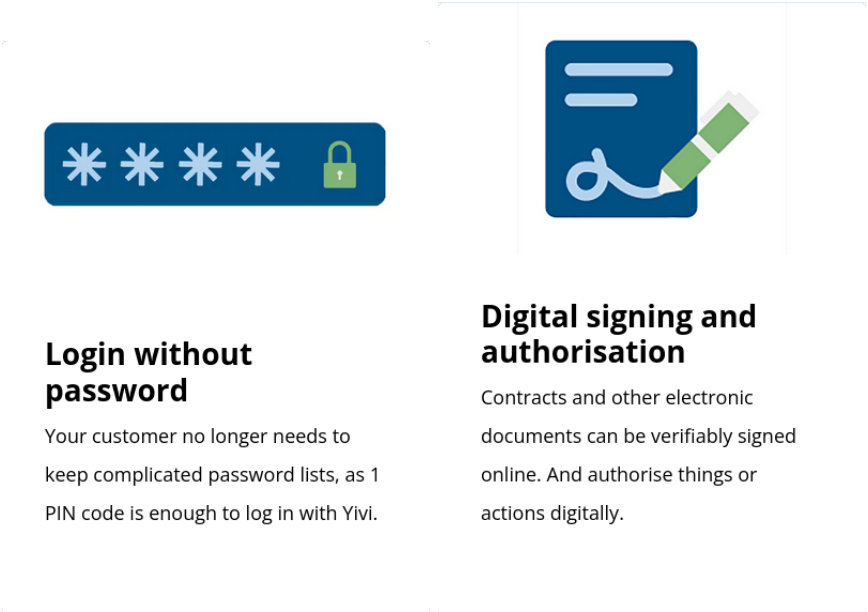
P4 **Selective disclosure**: Any subset of attributes from a credential can be revealed and proven independently.

P5 **Minimal information**: During verification protocols no other information is revealed to the verifier beyond the disclosed attributes, the credential names and the corresponding issuers.

2.4 A full-fledged ABC system - Yivi

Yivi[9] is an open-source application used in the real world!

Users are permitted to log in with their passwords to sign documents and contracts in various roles.



For example, one can sign up as a doctor or as a citizen.

This open-source solution is employed in several contexts. In the Netherlands, Yivi is used by the Chamber of Commerce, healthcare institutions (for electronic patient records), municipalities (such as in Amsterdam), universities (through SURF), and insurance companies, among others.

3 Why privacy matters

3.1 Backdoors and surveillance

Encryption backdoors refer to hidden methods of bypassing encryption, which can undermine the security of a system. They might imply that something which appears to be random is, in fact, not truly random. It is often difficult to determine whether such a weakness is intentional (as a backdoor) or simply the result of an unintentional vulnerability. This concept is also related to the idea of "front doors," which, like backdoors, allow unauthorized access. Whether it is a backdoor or a front door, the outcome is the same: the system is not safe anymore.

The issue is that some governments, including agencies such as the NSA and GCHQ, are advocating for backdoors in encryption systems to catch individuals engaged in illicit activities. However, this is a highly problematic approach. Introducing backdoors weakens cryptography not just for the intended targets, but for everyone. This also means that malicious actors could exploit these vulnerabilities. A key pitfall of this strategy is the intentional compromise of cryptographic systems, which undermines security for all users.

Here [10] some actual examples of Surveillance and Backdoors.

3.2 Privacy and Law

As Edward Snowden once stated,

"Arguing that you do not care about the right to privacy because you have nothing to hide is no different than saying you do not care about free speech because you have nothing to say."

[9] see https://www.yivi.app/en and https://privacybydesign.foundation/en/ for more detail

[10] https://www.schneier.com/essays/archives/2007/11/did_nsa_put_a_secret.html, https://www.theverge.com/2020/10/12/21513212/backdoor-encryption-access-us-canada-australia-new-zealand-uk-india-japan, http://dspace.mit.edu/handle/1721.1/97690

The issue is that privacy is not solely a personal matter; it also has broader societal implications. We do not live in an ideal world, and people have a fundamental need for privacy. Privacy is, in fact, a human right, as it is enshrined, for example, in Article 8 of the Charter of Fundamental Rights of the European Union.

According to Daniel J. Solove, privacy matters for the following reasons:

- Limit on Power: the more someone knows about us, the more power they can have over us.
- Respect for Individuals: if a person has a reasonable desire to keep something private, it is disrespectful to ignore that person's wishes without a compelling reason to do so.
- Reputation Management: how we are judged by others affects our opportunities, friendships, and overall well-being.
- Maintaining Appropriate Social Boundaries: we need places of solitude to retreat to, places where we are free of the gaze of others to relax and feel at ease.
- Trust: in relationships, whether personal, professional, governmental, or commercial, we depend upon trusting the other party.
- Control Over One's Life: without having knowledge of what data is being used, how it is being used, the ability to correct and amend it, we are virtually helpless in today's world.
- Freedom of Thought and Speech: we may want to criticize people we know to others yet not share that criticism of the world
- Freedom of Social and Political Activities: we protect privacy at the ballot because of the concern that failing to do so would chill people's voting for their true conscience.
- Ability to Change and Have Second Chances: there is a great value in the ability to have a second chance, to be able to move beyond a mistake, to be able to reinvent oneself.
- Not Having to Explain or Justify Oneself: it can be a heavy burden if we constantly have to wonder how everything we do will be perceived by others and have to be ready to explain.

An overlooked link between cryptography and privacy is law.

Privacy can be implemented through both architecture and policy. Architecture refers to the design of systems that provide privacy, such as Tor. However, implementing an effective privacy policy is challenging without the appropriate technical infrastructure in place. In the European Union, laws can encourage the use of encryption, such as the General Data Protection Regulation (GDPR). For instance, Article 32(1)(a) of the GDPR addresses the pseudonymization and encryption of personal data. Recently, the French Data Protection Authority (DPA) imposed a fine of €600,000 for the use of outdated encryption methods (EDF). Furthermore, Recital 83 of the GDPR encourages the application of security measures, including encryption, to protect personal data.

Cryptography is most effective when widely adopted, as it ensures protection for those who need it. For example, the use of Tor provides privacy for its users, and the widespread implementation of SSL/TLS protocols, particularly through initiatives like "Let's Encrypt," helps prevent eavesdropping and mass surveillance. If the entire web is encrypted, it becomes significantly more difficult to intercept data. Cryptography thus makes far more sense when applied at the societal level.

For instance, we rely on standards to ensure the security of cryptographic methods. The National Institute of Standards and Technology (NIST) conducts competitions to test new cryptographic algorithms, ensuring that they are robust and secure. Standards play a crucial role in reducing the risk of encryption backdoors.

Large-scale adoption of encryption can be driven by legal frameworks, such as the GDPR, and the imposition of regulatory requirements. Cryptography is a tool that requires broad adoption to be effective. This is why it must be incorporated into law, and why standards are essential in its implementation.

3.3 Ethics of cryptography

In 2015, the "FBI vs. Apple" case became a highly publicized legal and ethical debate. The case followed a tragic shooting in California, in which 14 people were killed and 22 injured. The shooters were killed by the police, who subsequently retrieved their phones. However, the phones were locked and designed to erase their data after 10 failed password attempts. The police, seeking to investigate the attackers' motives, requested that Apple create an update to bypass the phone's security features. Specifically, they asked Apple to disable the auto-erase function and to allow multiple password attempts remotely, without any delays. In response, Apple refused to comply, citing the potential risks of creating a weakened version of iOS. Apple argued that such a modification would create a backdoor into the system, not just for this particular case, but for every iPhone. The company contended that this would jeopardize the security of all users by exposing them to potential exploitation. The case also raised

concerns that there could be additional suspects involved and that creating such a backdoor could have far-reaching consequences for digital privacy and security. Ultimately, the conflict between national security interests and the protection of privacy was brought to the forefront of the public and legal discourse.

Cryptography has implicit ethical moral and policy dimensions. What does it mean that math has morals?

> *"Cryptography rearranges power: it configures who can do what, from what. This makes cryptography an inherently political tool, and it confers on the field an intrinsically moral dimension. "*
>
> Rogaway, The Moral Character of Cryptographic Work

The "crypto wars" date back to 30 years ago, when cryptography was considered a weapon, and as a result, the export of cryptographic technology from the United States was heavily restricted. At that time, cryptography was regarded with the same level of concern as firearms or explosives.

Should private communication be considered a right? If yes, should this right be absolute? What are the implications of such a moral right?

Cryptography alone cannot protect against business surveillance. Government surveillance is often closely linked with commercial surveillance, and state and corporate surveillance can be seen as two sides of the same coin. For example, Google Analytics was banned in the EU due to the lack of adequate data protection in the United States. Another notable case is the Cambridge Analytica scandal in 2016, where the company allegedly influenced the outcome of the US election by providing targeted political advertisements, which were deemed illegal. This could not have occurred without the vast amount of data provided by Facebook.

The key message is that while cryptography is an essential tool for protecting privacy, it is not a panacea. As with any tool, it requires careful handling and understanding. Cryptography is not the only means of safeguarding privacy—law and regulation also plays a crucial role in protecting individuals' rights.

TDA352/DIT352: Cryptography LP2, Fall 2024

Lecture Notes on

Commitments, Zero Knowledge Proofs

Lecturer: Elena Pagnin Lecture 11 on Dec 10

Scribes: Willem Brahmstaedt, Hanna Ek & Lucia Lavagnino Last update January 3, 2025

Contents

1 Verifiable Secret Sharing (VSS)

In the figure 1, Alice, acting as the dealer of a secret, divides the secret into several parts s_i, with each shareholder possessing one of these parts. Threshold cryptography makes it possible to reconstruct the secret using fewer than all the parts originally distributed.

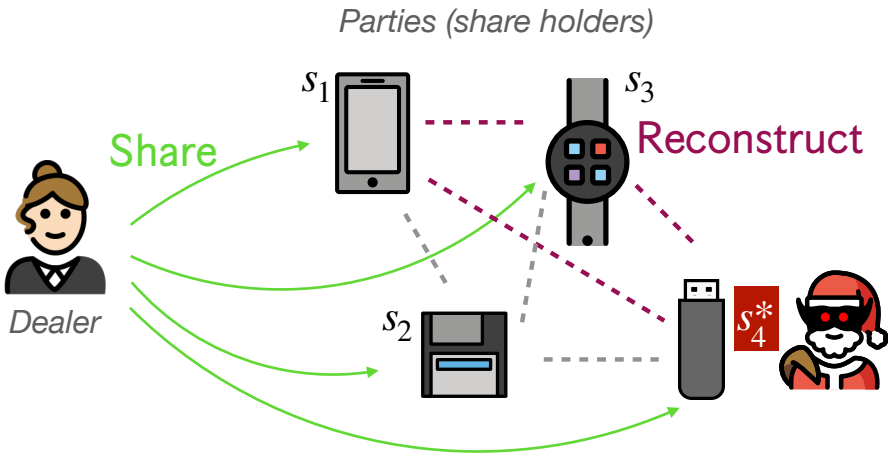


Figure 1: The adversary $\mathcal{A}$ changes the share from s_4 to s_4^*.

In this specific instance of figure 1, the dealer divides the secret into four shares, of which only three are required to reconstruct the original secret successfully.
The problem arises when one of the shares falls into the hands of an adversary. Even if the adversary cannot reconstruct the secret independently, what happens if they alter their share? How can we detect such a modification? The solution lies in Verifiable Secret Sharing (VSS). This approach ensures that it is possible to verify whether the correct shares are being used and that the reconstruction process leads to the correct secret.

1.1 A Simple Scheme

We will present a simple scheme to address this issue. The approach builds on any threshold secret sharing scheme, such as Shamir's scheme, under the assumption that only one share among the n shares is corrupted. The method involves running the Reconstruct algorithm on all possible subsets of shares of size t. The secret is then verified by

determining the most common value s among all the reconstructed results. This process ensures that the presence of a single corrupted share does not compromise the integrity of the reconstructed secret.

Note: The number of distinct subsets of size t in $\{1, 2, \ldots, n\}$ is given by $\binom{n}{t}$, where $\binom{n}{t}$ denotes the binomial coefficient, defined as follows:

$$\binom{n}{t} = \frac{n!}{t!(n-t)!}$$

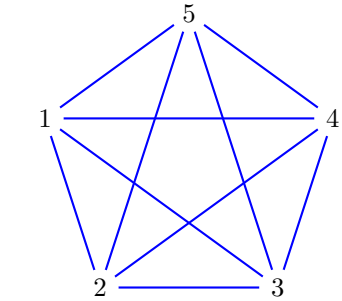
where

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases}$$

Example 1. With $n = 5, t = 2$, there are 10 distinct subsets in $\{1, 2, 3, 4, 5\}$ of size 2, indeed

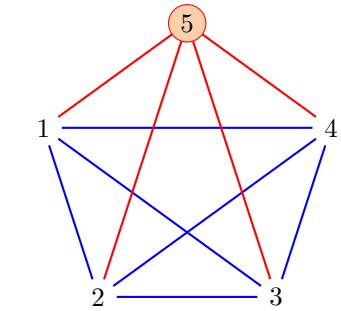
$$\binom{5}{2} = \frac{5!}{2!(5-2)!} = \frac{5 \cdot 4}{2} = 10$$

To gain intuition, we can examine the following graphical representation:



where each line represents a possible subset of two elements from the five available.

If there is one corrupted party (for example, party 5), there are 4 "bad" subsets that include the corrupted party, while there are 6 "good" subsets that contain only honest parties.



As demonstrated in the previous example, the underlying idea of this approach is that if there are more honest parties than dishonest ones, we can rely on a majority. This method works effectively when there is only one compromised device.
Hence, the method to verify the secret sharing involves running the reconstruction process on all possible sets and then observing the results.
This approach works, but it has several limitations. First, it is not efficient. Second, to verify the secret, more than t shares are needed. Finally, it only works when there are a few corrupted shares and many honest ones. To verify the secret s as the "most commonly reconstructed value", the number of correct reconstructions (those that do not involve any corrupted share) must be larger than the number of incorrect reconstructions (those that use at least one corrupted share). Let c denote the number of corrupted shares. The number of correct reconstructions is given by

$$V = \frac{(n-c)!}{t!(n-c-t)!}$$

and the number of incorrect reconstructions is

$$B = \sum_{i=1}^{c} \frac{(n-i)}{(t-i)!(n-i-t)!}.$$

1.2 Shamir & Pedersen

Pedersen Verifiable Secret Sharing is a method that combines Shamir's Secret Sharing with Pedersen Commitment. The intuition behind this approach is as follows: each time the dealer distributes a share, she also publishes a commitment to the share. Now, if any share holder were to alter their share this would be detected as they modified shared is no longer an opening of the commitment of the original share. This forces the parties to employ the correct shares when running reconstruction. Additionally, by committing to the secret itself, the hiding property of the commitment ensures that the secret cannot be deduced from the commitment. The binding property ensures that the shares will only match if the correct secret is reconstructed.

Example 2. Pedersen Verifiable Secret Sharing (VSS) is built by the following algorithms:

- $\mathsf{Share}(s)$: Sample at random $a_1, \ldots, a_{t-1}, b_0, \ldots, b_t - 1 \xleftarrow{\$} \mathbb{Z}_p$, with $a_{t-1} \neq 0, b_{t-1} \neq 0$.
  Define the polynomials: $a(x) = s + a_1 x + \ldots + a_{t-1} x^{t-1}$, $b(x) = b_0 + b_1 x + \ldots + b_{t-1} x^{t-1}$
  Compute the share for party P_i as: $s_i = (a(i), b(i))$.
  Publish the set $\{C_i\}_{i=0}^{t-1}$ of commitments to each share: $C_i = g^{a_i} h^{b_i}$, where $a_0 = s$.
- $Ver(s_i, \{C_j\})$: Check share consistency $g^{s_i[1]} \cdot h^{s_i[2]} == \prod_{j=0}^{t-1} (C_j)^{i^j}$
- $\mathsf{Reconstruct}(\{s_{i_1}, \ldots, s_{i_t}\}, \{C_{i_1}, \ldots, C_{i_t}\})$: do like in Shamir secret sharing to reconstruct $a(0) = s$ and $b(0) = b_0$ and verify the consistency $g^s \cdot h^{b_0} == C_0$.

Note: the Veralgorithm works since

$$g^{s_i[1]} \cdot h^{s_i[2]} = g^{a(i)} \cdot h^{b(i)} = g^{\sum_{j=0}^{t-1} a_j i^j} \cdot h^{\sum_{j=0}^{t-1} b_j i^j} = \prod_{j=0}^{t-1} (g^{a_j} \cdot h^{b_j})^{i^j} = \prod_{j=0}^{t-1} (C_j)^{i^j}$$

We use the second polynomial $b(x)$ to introduce randomness, which enables us to commit securely.

2 Zero-Knowledge Proofs (ZK)

2.1 Introduction

In cryptography, one can have the cake and eat it too! That is, you can prove that you have a secret, without giving any information apart from the fact that you know it. This is the basis for Zero Knowledge Proofs [1].

The concept underlying Zero-Knowledge Proof is quite unconventional: how can one prove something without disclosing any knowledge? For instance, it is possible to demonstrate knowledge of the correct secret key without actually revealing the key itself.

How can we formalize Zero-Knowledge Proofs (ZKP) using mathematics and cryptography? Let us consider another example to illustrate the concept. Imagine a Sudoku puzzle—how can I convince you that the puzzle is solvable? A straightforward approach would be to solve it and share the solution. However, the goal is to prove that I know the solution without revealing what it is.

In the accompanying figure 2, x represents the statement, which is public and known to everyone (in this example, the Sudoku puzzle). On the other hand, w is the witness, a piece of information that proves the validity of the statement (in this case, the solution to the Sudoku puzzle).

[1] Here https://youtu.be/0pu8tviH2sc?t=68 is a video that provides an intuitive explanation of the concept behind Zero-Knowledge Proofs.

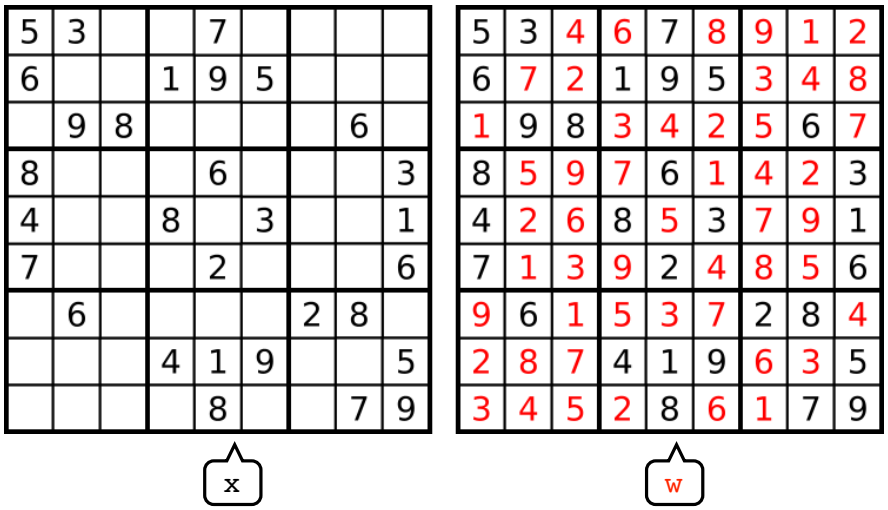


Figure 2

Mathematically, for the Sudoku example, we can define L as the set of all solvable Sudoku puzzles. In the most general formalisation, the Zero-Knowledge Proof would demonstrate that

$$x \in L \qquad iff \qquad \exists w \, s.t. \, R(x, w) = 1$$

hence x is in the set L if and only if there exists a witness w such that the relation R is satisfy, i.e. $R(x, w) = 1$.

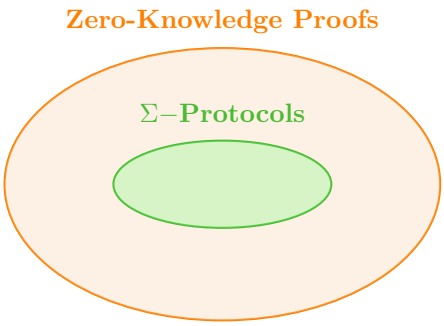
Definition 1. A **zero-knowledge (ZK) proof** is an interactive protocol, run between two parties called Prover (P) and Verifier (V), in which the Prover probabilistically convinces the Verifier of the correctness of a given mathematical proposition, and the Verifier learns nothing beyond the truth of the statement.

The protocol is described as interactive because it involves a dynamic exchange of information, where one party sends a message, receives a response, and so forth. It is not merely a straightforward back-and-forth communication. The prover is the individual who seeks to demonstrate that he possesses knowledge of the witness. In Zero-Knowledge Proofs, the prover is probabilistic, meaning they utilize randomness in their actions to convince the verifier, with a certain level of probability, of the validity of a given mathematical proposition. The verifier should be convinced with a certain probability of the truth of the statement, while gaining no additional knowledge beyond the validity of the statement itself.

Example 3. Here, we will present two examples of the formalisation of Zero-Knowledge Proofs, each based on a fundamental mathematical concept: the discrete logarithm and factorization.

	dLog	**Factoring**
Relation $R(x, w) = 1$ if and only if	$x = g^w \in \mathbb{G}$	$x = w[1]w[2] \wedge w[1], w[2]$ are primes
Statement x	$x = \mathsf{pk}$	$x = N$
Witness w	$w = \mathsf{sk}$	$w = (p, q)$

The theory of Zero-Knowledge Proofs (ZK Proofs) is both deeply fascinating and fundamental to the field of cryptography. ZK Proofs play a crucial role in achieving malicious security within multi-party computation protocols, ensuring that even adversarial participants cannot compromise the protocol's integrity. Despite their utility, ZK Proofs are generally computationally and communicatively expensive, requiring significant resources to implement. Nevertheless, the theoretical framework of zero-knowledge proofs are elegant and rich, forming a cornerstone of modern cryptography. In the context of cryptographic protocols, ZK Proofs are instrumental in enforcing adherence to the protocol. They enable parties to demonstrate that they have executed the protocol correctly without revealing any additional information, thereby ensuring compliance while preserving privacy. The world of Zero-Knowledge Proofs is vast and highly fascinating; however, we will focus exclusively on Sigma protocols.



2.2 Σ−Protocoll

The name "Sigma protocol" derives from the "shape" of the protocol, as the exchanges in the interactive protocol resemble the Greek letter sigma Σ.

Definition 2. A **Sigma-protocol** for relation R is a *three-move, public coin protocol* of the form in Figure 3 that satisfies the following below properties. (public coin means that e is sampled at random).

- **Completeness:** If the Prover (P) and Verifier (V) follow the protocol on input x and private input w (known by P) where $(x, w) \in R$ then V always accepts.
- **Special soundness:** There exists a PPT algorithm $\mathcal{E}$ (extractor) that given any x and any pair of accepting transcripts $(a, e, z), (a, e', z')$ for x with $e \neq e'$ outputs w such that $R(x, w) = 1$.
- **Special honest verifier zero knowledge (SHVZK):** for every x and w such that $R(x, w) = 1$ and every $e \in \{0,1\}^t$ there exists a PPT algorithm Sim (simulator) which given input (x, e) outputs transcripts (a, e, z) that are distributed like real conversations:

$$\{\mathsf{Sim}(x, e)\} =_c \{\langle P(x, w), V(x, e)\rangle\}.$$

Figure 3: Σ−Protocoll

It is a three-move protocol, where the prover first sends a to the verifier, the verifier then responds with e, and finally, the prover sends z back. In the literature, a is typically referred to as the commitment, e is the challenge, and z is the response. At the end of the protocol, the verifier must decide whether to accept or reject the statement the prover is attempting to prove. Remark that (a, e, z) is accepted by the Verifier and the challenge e is known in advance.

Intuition of the properties of the Σ-Protocol.

- **Completeness:** The protocol "completes" (ends) correctly, i.e., the Verifier will always accept an honest Prover.
- **Special Soundness:** This property protects the Verifier, i.e., if the Verifier accepts it must be the case that the prover knows the witness (with large enough probability). With this property, we are saying that if the prover is careless enough to reuse the same a in two different instances, it becomes possible to extract the secret witness. Thus, soundness ensures that there is a method to extract the witness from the prover. In

other words, the prover must indeed know the witness for the protocol to hold true.
An honest verifier accepts the proof of a dishonest prover for a false statement with a probability not greater than a certain bound (e.g., $\frac{1}{|e|}$). This means the Prover must know w, otherwise she will not pass the verification check (unless with probability $\frac{1}{|e|}$). This ensures that the Verifier accepts only honest provers (who know w), and rejects malicious provers (who pretend to know w, but do not actually know it) with overwhelming probability. To prove this we need to show that it is possible to "extract" the w somehow. We cannot hope to extract w from a single transcript (otherwise there would be no zero knowledge, no simulator), but special soundness guarantees that we can extract w from 2 transcripts that share the same initial input a. Having two such transcripts might look artificial, but it is a simple example of an intrigued proof technique called "rewinding argument" (which we do not cover in this course). To give you the gist of it, special soundness gives us a logical reasoning (algorithm) for extracting the witness, the existence of the extractor algorithm ensures that the Prover must know w (with overwhelming probability); and the extraction works independently of how realistic it is that the prover will indeed use the same a for two runs of the same sigma protocol.

- **SHVZK:** This property protects the Prover, i.e., a transcript does not leak the witness (since the transcript could be efficiently simulated)
  The verifier learns nothing else about the honest prover's private input from the protocol execution, other than the fact that the claimed statement is true. This ensures that the Prover's w is not leaked to the verifier throughout the interaction. The way to formally prove that a transcript (a, e, z) does not leak any information about the witness z, is to show that it is possible to generate a "good looking" (i.e., accepting) transcript even without the knowledge of the witness. However, if this was always possible, then we would lose the meaning of "proof of knowledge". So we can be able to generate an accepting transcripts if we either know w, or we can choose the challenge e. The simulator, therefore, gets e as input and then produces (a, e, z) that has the same distribution as an honest transcript but was produced without w. This means that the transcript per se does not leak any information about w (but this Simulator only works if e is known, hence it is in the Verifier's interest to pick the challenge e at random, or at least in a way that is not predictable by the prover).

It is in the Prover's interest not to choose the same randomness r (and thus a) for two protocol runs, as doing so would allow us to extract the witness. However, the fact that we can mathematically extract w from two distinct conversations that start with the same a ensures that only a Prover who knows w can respond correctly to any challenge. Similarly, it is in the Verifier's interest to choose the challenge randomly and only after receiving a from the Prover. Moreover, the fact that knowing e in advance provides us with a mathematical way to simulate accepting conversations guarantees that a transcript transfers "zero knowledge" about the witness.

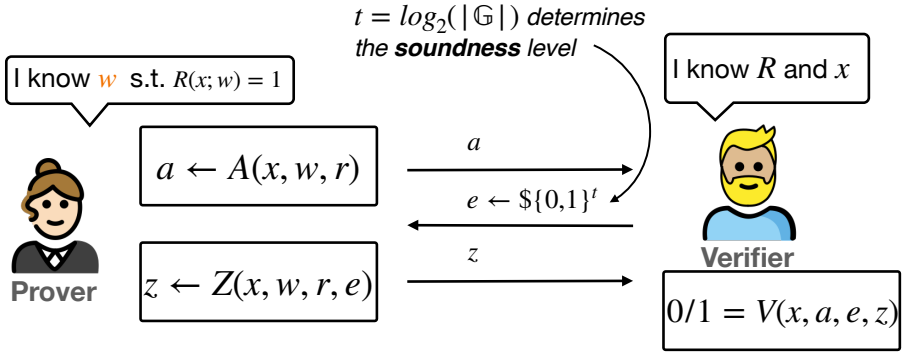


Figure 4: Generic form of a $\Sigma-$protocol

Every Sigma protocol can be expressed using three algorithms (as in figure 4): A, Z, and V. The algorithm A takes the public state commitment x, the secret witness w, and some randomness r, and produces a. The challenge e is always chosen randomly. The response algorithm Z, which takes the public state commitment x, the secret witness w, some randomness r, and the challenge e, outputs the answer z. The verifier uses the verification algorithm V, which takes as input the public state commitment x, a, the challenge e, and the answer z, and outputs either 0 or 1. Therefore, when considering a Sigma protocol, we only need to focus on the three algorithms: A, Z, and V.

2.3 Schnorr $\Sigma-$protocol (aka Schnorr identification)

Here is an example of $\Sigma-$protocol. The prover aims to convince the verifier that she knows the private key corresponding to the given public key.

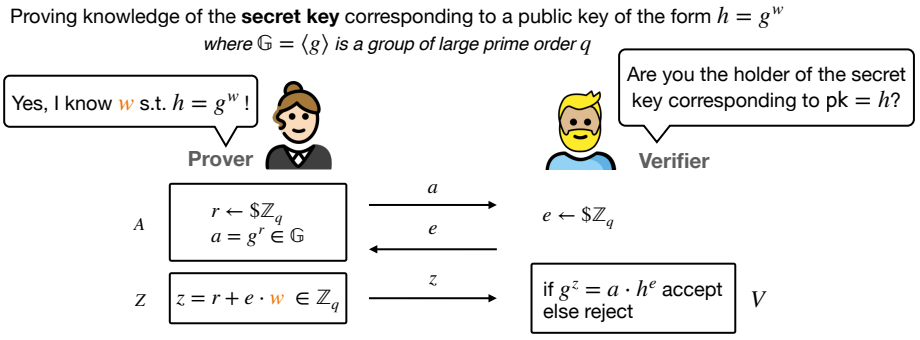


Figure 5: Schnorr $\Sigma-$protocol.

In the Schnorr identification protocol, we have that

$$A : r \xleftarrow{\$} \mathbb{Z}_q, a = g^r \in \mathbb{G}. \qquad Z : z = r + e \cdot w \in \mathbb{Z}_q. \qquad V : g^z = a \cdot h^e.$$

This protocol satisfies all three properties of Zero-Knowledge Proofs:

Completeness: If the prover knows w, the verifier will accept the proof. Indeed, if the prover behaves correctly, we have: $z = r + e \cdot w$ and $g^z = a \cdot h^e$, hence

$$g^z = g^{r+e \cdot w} = g^r \cdot (g^w)^e = a \cdot h^e$$

Special Soundness: Suppose we have two transcripts (a, e, z) and (a, e', z'), then it is possible to extract the witness from them:

$$\begin{cases} z &= r + e \cdot w \mod q \\ z' &= r + e' \cdot w \mod q \end{cases}$$

$$z' - z = (e' - e) \cdot w \mod q$$

and finally we can recover the correct witness as $w = (z' - z)(e' - e)^{-1} \mod q$.

SHVZK: If the verifier reveals the challenge, then anyone can pretend to be the prover. In order to prove SHVZK, we could to construct the simulator $\mathsf{Sim}(e) \to (\tilde{a}, e, \tilde{z})$ by sampling $\tilde{z} \xleftarrow{\$} \mathbb{Z}_q$, and then compute $\tilde{a}$ as $\tilde{a} = g^{\tilde{z}} \cdot h^{-e}$.

2.4 Proving Knowledge of Pedersen Commitments

We have just demonstrated how to prove knowledge of the secret key corresponding to a Diffie-Hellman public key. Similarly, we can prove knowledge of the Pedersen commitment, which means that we know the message committed to a specific Pedersen commitment. In other words, we can prove that we know the message that is committed to that particular Pedersen commitment.

Recall the Pedersen commitment:

Definition 3. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q, and h be a random element in $\mathbb{G} \backslash \langle g \rangle$. Let q, g, h be public information. The **Pedersen** commitment function is defined as:

$$\mathsf{Commit}(m, r) = g^m h^r = c \text{ for } r \xleftarrow{\$} \mathbb{Z}_q.$$

In the Pedersen commitment identification protocol we want to prove knowledge of m, r such that $c = (g^m \cdot h^r)$. That means in the terminology introduced that (m, r) is the witness, c is the statement and $c = g^m \cdot h^r$ is the relation.

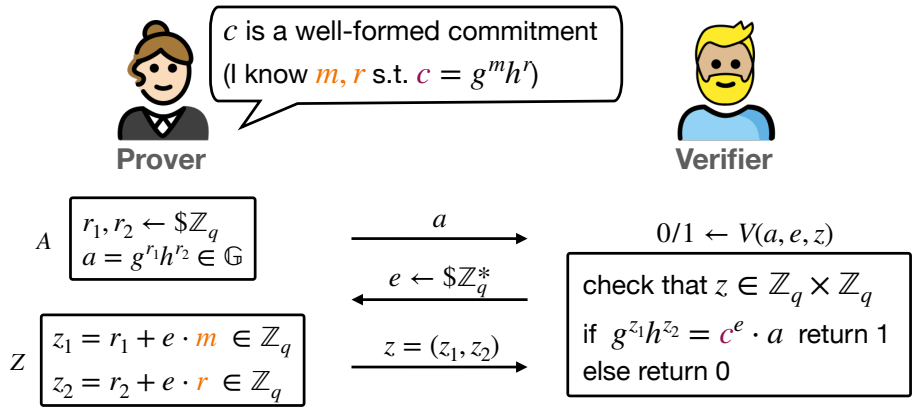


Figure 6: Proving Knowledge of Pedersen Commitments

Remark: The a has the same structure as in the Schnorr Σ-protocol discussed earlier. The construction is quite similar to the previous one; however, in this case, we are convincing the verifier about the message rather than the secret key. We have that the algorithms A, Z, V are as follows:

$$A : r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q, a = g^{r_1} h^{r_2}. \qquad Z : \begin{cases} z_1 & = r_1 + e \cdot m \\ z_2 & = r_2 + e \cdot r \end{cases} \qquad V : g^{z_1} \cdot h^{z_2} = c^e \cdot a.$$

This protocol satisfies all three properties of Zero-Knowledge Proofs, for example we can see that the Completeness property is satisfied:

Completeness: If the prover knows w, the verifier will accept the proof. Indeed, if the prover behaves correctly, we have: $\begin{cases} z_1 & = r_1 + e \cdot m \\ z_2 & = r_2 + e \cdot r \end{cases}$, hence

$$g^{z_1} \cdot h^{z_2} = g^{r_1 + e \cdot m} \cdot h^{r_2 + e \cdot r} = g^{e \cdot m} \cdot h^{e \cdot r} \cdot (g^{r_1} \cdot h^{r_2}) = (g^m \cdot h^r)^e \cdot (g^{r_1} \cdot h^{r_2}) = c^e \cdot a$$

3 Removing Interaction (NIZK)

So far, with the $\Sigma-$protocol, we have always seen Alice and Bob interacting with each other. But what if we remove the interaction? And why do we want a non-interactive ZK protocol? In the interactive setting only the person who sent the challenge e can verify that e was sent after receiving a and that the correct e was used. This means that the protocol cannot men publically verified.

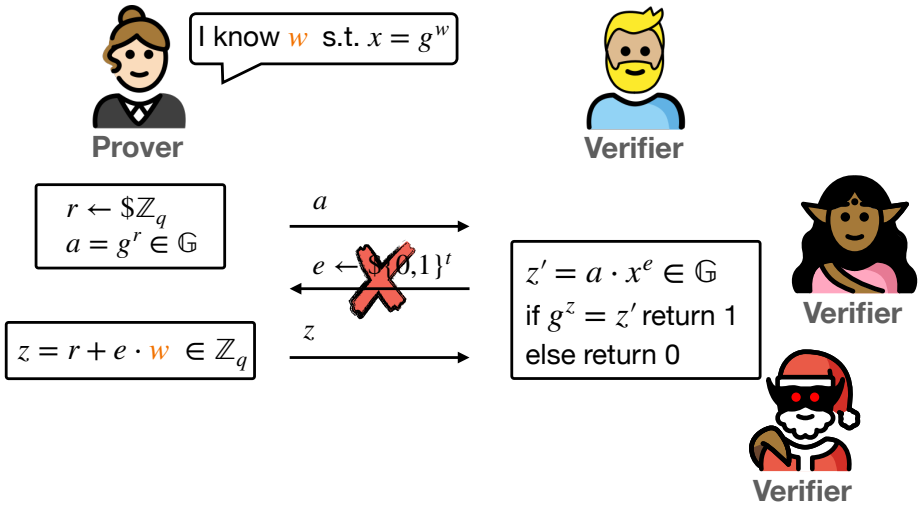


Figure 7: Schnorr $\Sigma-$Protocol NON-interactive

The question is: How can we generate a valid challenge if it is not provided by the verifier? The solution is called the Fiat-Shamir Heuristic.

3.1 Fiat-Shamir Heuristic

Intuitively, why does it work? It works because, given the input to the hash function, you can compute the output, but you cannot influence it. The result is that e no longer comes directly from the verifier; instead, e is derived from the hash function. Fiat-Shamir Heuristic model the hash function as a random oracle and computes the challenge as $e = H(g, x, a)$.

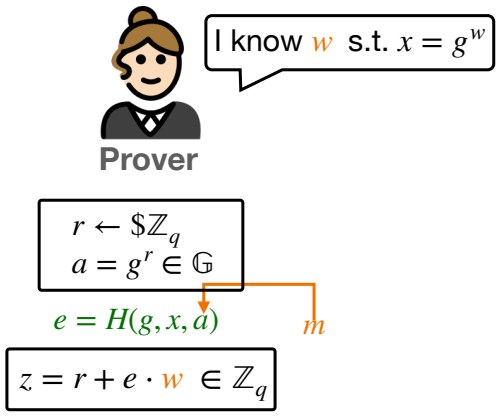
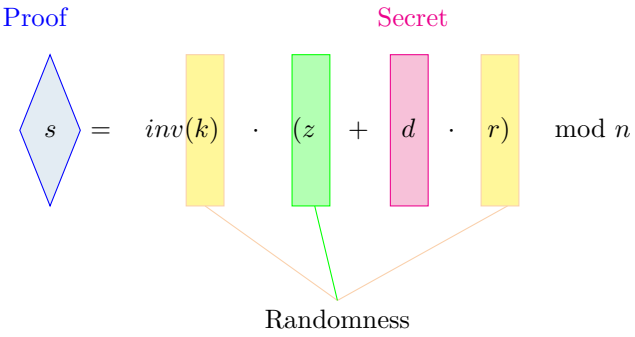


Figure 8: Fiat-Shamir Heuristic

We remark on similarity between non-interactive Schnorr Σ-protocols and digital signatures. If we think instead of picking r at random let $r = m$, the protocol can be transformed into a digital signature protocol.

Recipe to create a digital signature from a ZK Proof

1. Pick randomness r, k ;
2. generate new (unpredictable) randomness z using the hash function;
3. use the secret s and hide it with both of the randomnesses;
4. return a proof s of knowledge of the secret value.



This is the same procedure as ECDSA is built.